175 PTAS



CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen IX-Fascículo 101

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-181-X (tomo 9)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 258512
Impreso en España-Printed in Spain-Diciembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.



El poder del CAD

El diseño asistido por ordenador (CAD) ya se utiliza ampliamente en la industria. Examinemos esta eficaz técnica

La gran diferencia que distingue a los gráficos por ordenador elementales del auténtico CAD radica en que este último está relacionado con mucho más que el mero dibujo del objeto. Los sistemas CAD más sofisticados poseen un sentido real del objeto alojado en sus memorias y no una simple imagen del mismo. Algunos sistemas son capaces de probar el objeto, montando los componentes, simulando tensiones y fuerzas y asegurando que el producto final no se derrumbe una vez construido.

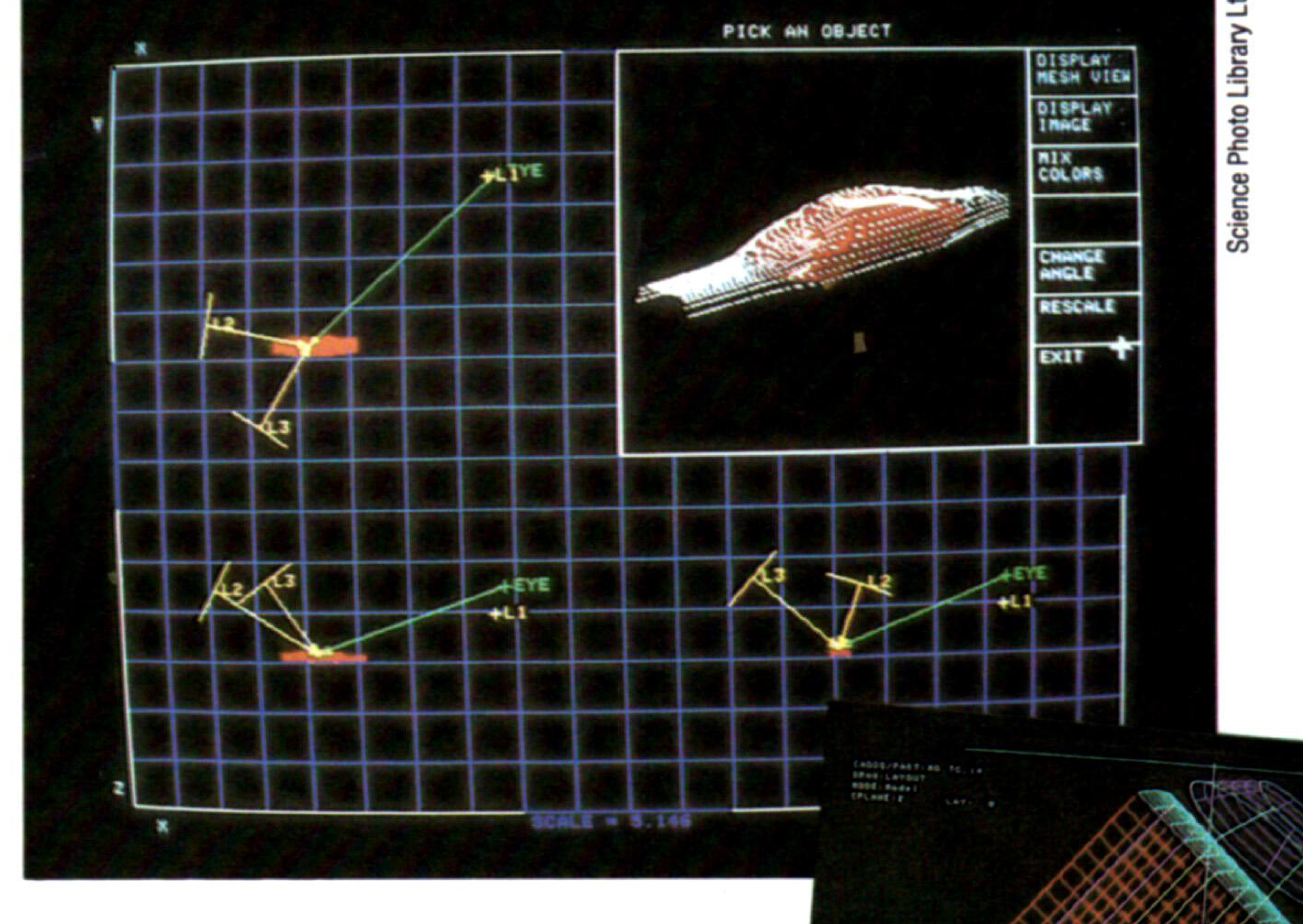
Las imágenes de la mayoría de los sistemas CAD se crean en realidad de forma diferente a las de los sistemas gráficos normales. Los gráficos por ordenador comunes construyen las imágenes con pixels, pero éstos no son lo suficientemente precisos para el CAD, que utiliza, en cambio, el sistema de gráficos de vectores. Éste basa el dibujo en la descripción matemática de las formas. Por lo tanto, en los gráficos por pixels una línea sería una fila de puntos en la pantalla, mientras que en un gráfico por vectores se entendería como las coordenadas de ambos extremos y la distancia más corta entre ambos. Los gráficos por vectores son más exactos que los gráficos por pixels, permitiendo que el procesador los trate con resoluciones mucho mayores de las que puede visualizar el monitor.

Los sistemas CAD más sencillos dibujan en dos dimensiones y en ocasiones pueden añadir la apariencia de una tercera, cosa que generalmente se conoce como dibujo 2 1/2-D (en dos dimensiones y media). Tal sistema puede satisfacer las necesidades de un estudio de arquitectura, donde la «media» dimensión adicional permite que el arquitecto muestre un alzado además de una planta.

Las empresas que aplican el CAD para diseñar sistemas de circuitos electrónicos también utilizan sistemas 2-D, pero de mayor complejidad. Debido a que los sistemas CAD pueden retener información no sólo acerca de un dibujo sino también acerca del producto real, en el campo de la electrónica, por lo general el CAD va acoplado con programas de prueba; una vez que el diseñador de un chip ha elaborado el mapa de los millones de circuitos de un microprocesador, el sistema comprobará que el conjunto de circuitos funcione correctamente.

El uso más complejo del CAD concierne al diseño de objetos para su fabricación. Requiere 3 dimensiones y, por tanto, necesita más memoria y un procesamiento más rápido con el fin de tratar la imagen. Dado que los diseños se han de convertir en productos acabados, ha de ser muy preciso.

El diseñador que utiliza un sistema CAD primero elabora los mapas de los componentes individuales (cada uno de ellos a una escala específica) y luego los almacena. A medida que el trabajo avanza, los distintos elementos se pueden recuperar de la base de datos e incluirse en el dibujo.



Science Photo Library Ltd.

Anatomía de una imagen
El CAD aplica la tecnología de gráficos por vectores para manipular imágenes y visualizarlas en muy alta resolución, como podemos ver en esta instantánea de pantalla de un ensamblaje por soldadura diseñado utilizando técnicas CAD. Los sistemas avanzados, sin embargo, también permiten probar los diseños antes de su fabricación. En la imagen inferior, un superordenador Cray comprueba la aerodinámica de un nuevo vehículo en la General Motors

Los dibujos 3-D de sistemas CAD vienen en diversas formas, de las cuales la más simple es el Wireframe (entramado). Éste presenta el objeto en tres dimensiones en el espacio, pero sólo en boceto. No hay ninguna imagen de las superficies. La 3-D completa y el modelado de sólidos proporcionan las superficies, y es el sistema en sí mismo el que tiene un sentido del objeto tal como es en realidad.

Los sistemas más complejos pueden producir imágenes coloreadas, con hasta 256 colores diferentes en pantalla al mismo tiempo. El sombreado suele añadir profundidad y contribuye a ofrecer una imagen más realista.

Los usuarios de CAD tienen a su disposición una gama de efectos cuando forman sus imágenes, además de la capacidad para manipular formas geométricas. He aquí una relación de los más importantes:

Dimensionado: Una vez terminado el dibujo, el diseñador puede hacer que el sistema añada automáticamente todas las cotas de las medidas y lo deje preparado para dibujar. Dado que el sistema retiene información en forma numérica, ésta es una operación directa.

Plumeado: Los diseñadores necesitan realzar secciones. El plumeado es una técnica de sombreado automático que utilizan los artistas técnicos.

Estratificación: Los diseños se pueden formar por partes. Quizá un arquitecto desee conservar la planta de una casa en una parte y los desagües y tuberías en otra. Ambos podrían aparecer en la misma salida impresa o bien por separado, según cual sea la información que se necesite.

Segmentos: En ocasiones se denominan *entidades* y son los elementos individuales que componen un dibujo. Se retienen como una especie de catálogo de dibujos ya hechos que puede reclamar el diseñador.

Banda elástica: Esta técnica permite que el diseñador estire las líneas o las acorte (como si fueran

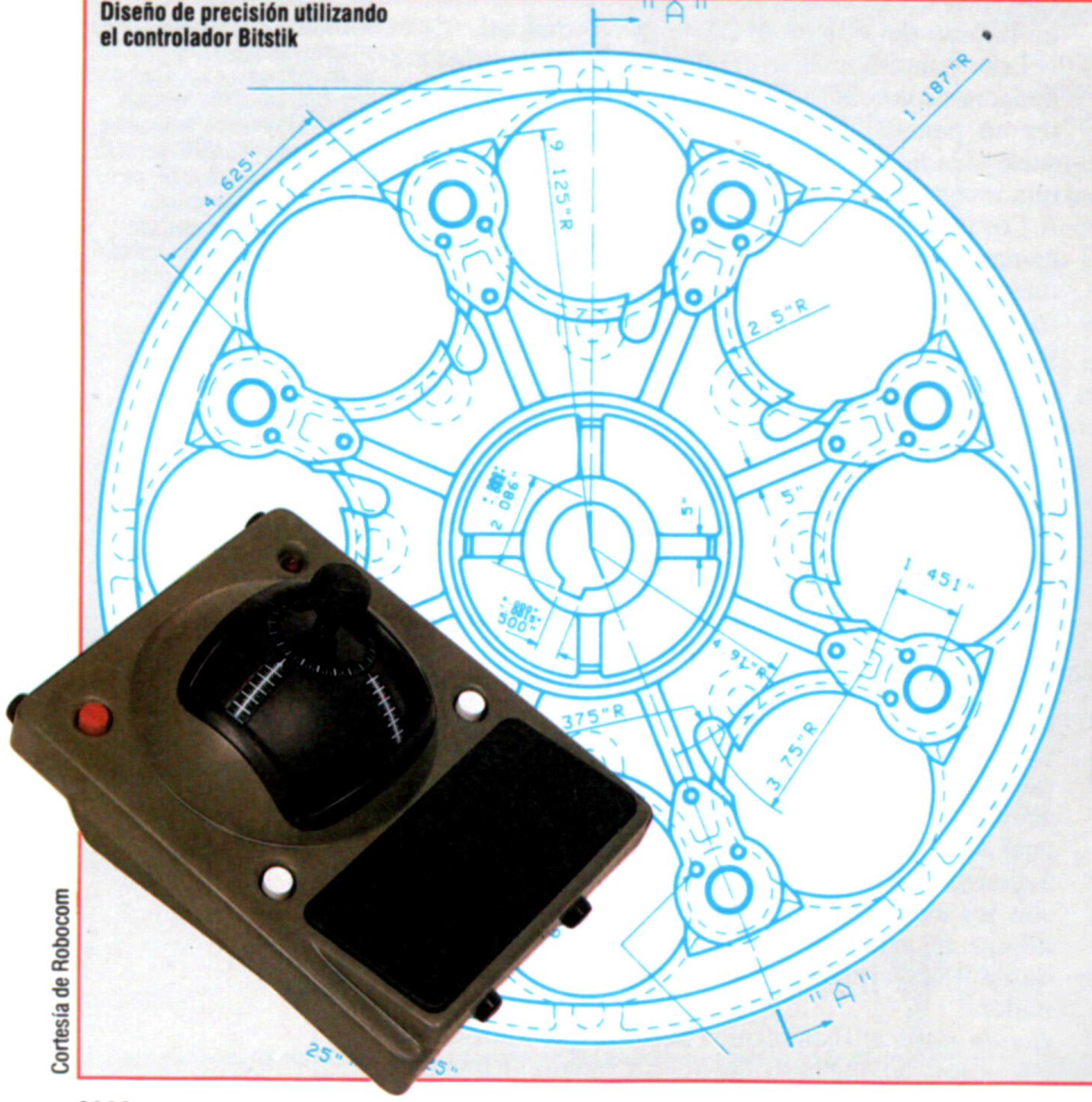
elásticas) para colocarlas allí donde las necesita.
Escalado: Permite que el diseñador dibuje en el tamaño que desee y después almacene la información a escala con el resto del dibujo.
"Zoom" y "pan": El primero de estos dos términos cinematográficos define la acción de enfocar una sección del dibujo y ampliarla hasta llenar toda la pantalla. El *pan* permite al diseñador volver hacia atrás, como estaba antes, y contemplar el dibujo en su totalidad.

Los usuarios de CAD tienen una gama de dispositivos que pueden emplear según el tipo de trabajo que estén realizando. Entre los más corrientes están la tablilla digitalizadora, el lápiz óptico, la bola seguidora y el ratón.

Los procesadores de imágenes se utilizan para diseños sumamente complejos y fotografías. Trabajan fotografiando la imagen a introducir y descomponiendo la fotografía en los elementos sombreados que componen una imagen de pantalla.

El teclado se emplea frecuentemente para introducir información en sistemas CAD, puesto que es una forma de gran precisión para entrar las coordenadas aplicadas en los gráficos de vectores.

En CAD también se utilizan palancas de mando, pero son mucho más sofisticadas que las que se emplean para los juegos. El Bitstik, de Robocom, posee potenciómetros de precisión que miden el movimiento de la palanca con seis lugares decimales. La palanca controla un cursor en la pantalla y, mediante pulsadores situados en la caja, se puede utilizar para seleccionar funciones del menú. El extremo del brazo del Bitstik se puede regular para efectuar un *zoom* o un *pan*.



Diseño de precisión utilizando el controlador Bitstik

Cortesía de Robocom

CAD en el micro

El diseño asistido por ordenador (CAD) está ganando creciente popularidad entre los diseñadores como herramienta fiable y eficaz desde el punto de vista del costo, para ayudar a producir diseños originales, o bien para probar posibles diseños sin necesidad de construir costosos modelos. Aunque el CAD requiere una gran cantidad de equipos y periféricos para obtener resultados profesionales, muchos de los sistemas que existen actualmente se basan en micros personales bastante modestos. El sistema HRX, para el ordenador Memotech 512, es un paquete que permite mejorar con ordenador las pantallas obtenidas con una cámara de video. Los "fotogramas" de la cámara de video pueden ser leídos por el sistema de ordenador y se pueden obtener numerosos efectos. Entre éstos se incluye la posibilidad de realizar *zooms*, contraer o hacer rotar la imagen. También se pueden cambiar completamente los colores aplicados a la imagen. El sistema de color que utiliza el formato HRX emplea los tres colores básicos de un sistema de video (rojo, verde y azul) y tiene una capacidad de más de 16,5 millones de colores por pixel. Un aparato de televisión normal sólo requiere una capacidad de color de 250 000. Por supuesto, para funcionar eficazmente, un sistema de este tipo requiere muchísimo hardware adicional. Con la necesidad de procesar tanta información, es obvio que se requiere una gran cantidad de memoria. El usuario, en consecuencia, necesita un controlador de doble anchura y tarjeta de memoria, con tarjetas de memoria adicionales si se ha de añadir color. Además, se requiere un convertidor A/D *flash* especial, simple (monocromático) o de tres canales (color) para aceptar información proveniente del ordenador MTX. Un sistema completamente equipado almacenará los datos de una cámara de video a la velocidad de 7,2 Mbytes por segundo. Obviamente, un sistema que ofrezca tal potencia es bastante caro. Mucho más asequible es el sistema Bitstik, para el BBC Micro. Este paquete permite que la máquina produzca dibujos técnicos de calidad profesional. El sistema consta de un controlador Bitstik (una especie de palanca de mando de gran precisión), un disco de sistemas, una ROM de aplicaciones de 8 K y un disco biblioteca que contiene numerosos tipos y símbolos comunes. Sin embargo, para poder operar el sistema, también se ha de dotar al micro de un segundo procesador 6502 y de unidades de disco gemelas de 80 pistas. Para poder sacar el máximo partido del sistema, el usuario también necesita un monitor en color y un plotter en color. De esta manera, el usuario obtiene un sistema que, según afirman los fabricantes, permite que alguien que carezca de formación artística produzca diseños gráficos estándares a nivel profesional. La base del sistema es la biblioteca de formas contenidas en disco y las "primitivas", como líneas y arcos, contenidas en ROM. Los diagramas se forman construyendo diseños más complejos a partir de estas primitivas, que luego se pueden incorporar a diseños mayores. La parte central del sistema es el propio controlador Bitstik, que permite posicionar las primitivas con gran precisión

Haciendo planos en etapas simples

Diseño escenográfico utilizando el Matchbox

Dibujo arquitectónico detallado producido con el Matchbox

Si bien el CAD suele asociarse al desarrollo de productos industriales, como automóviles, las técnicas utilizadas se pueden adaptar a casi cualquier aplicación en la que se requiera "modelado". Uno de los usos que se ha dado al CAD en un entorno no industrial es en el teatro. Una de las principales preocupaciones del escenógrafo es disponer la escenografía para que produzca el efecto visual requerido, sin entorpecer la visión del público. El problema es complicado para las producciones itinerantes que van de teatro en teatro, cada uno de los cuales tiene su propio "punto de vista". Tradicionalmente este problema se ha resuelto mediante la creación de modelos del teatro con reproducciones a escala de la escenografía, que se desplaza a través del escenario hasta hallar la disposición óptima. Sin embargo, un equipo de diseñadores de teatro del Theatre Royal, Stratford, en el East End de Londres, ha colaborado con los asesores de diseño de Robary Ltd. para producir el Matchbox, un paquete que permite crear escenografías en una VDU y contemplarlas desde cualquier ángulo dentro de cualquier teatro determinado. Las ventajas del sistema son que un diseñador puede crear una escenografía y pasarla por todas las vistas y teatros en los que se utilizará. Si la escenografía no se adapta a un teatro dado, se puede modificar, desplazando la utilería por el escenario de forma instantánea, en vez de tener que volver a construir los modelos a partir de cero. Además, el sistema Matchbox proporciona planos y dibujos de precisión, a partir de los cuales pueden trabajar los diseñadores y constructores

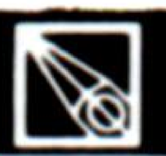
Theatre Royal, Stratford East

Hardware CAD



No es sólo el software lo que se elabora a la medida para su empleo en aplicaciones CAD. Un indicio de la importancia que se le otorga al diseño asistido por ordenador es el hecho de que también se están desarrollando algunos ordenadores que le otorgan preeminencia a las necesidades del CAD. El MG-1 Workstation, de Whitechapel Computer Works, es un miniordenador que contiene numerosas configuraciones esenciales para el procesamiento eficaz del CAD. El MG-1 se basa en el procesador de 16 bits National Semiconductor NS32016, que proporciona el tipo de potencia de procesamiento veloz que requiere el CAD. La visualización de gráficos tiene una resolución de 1024 × 800 pixels y, con el fin de producir imágenes rápidas sin parpadeo, incluye numerosas características avanzadas. La máquina está equipada con refresco de memoria directo, con lo que se evita la necesidad de tampones de fotogramas, que reducen la velocidad a la cual se refresca la pantalla VDU. Además, con el objeto de reducir el problema del movimiento desigual del cursor con el ratón, hay un procesador separado para manipular el ratón y la entrada por teclado. El MG-1 opera bajo el GENIX. Ésta es la versión de National Semiconductor del popular OS multitareas Unix. Se han desarrollado numerosos paquetes para correr en el MG-1 bajo este sistema. Por ejemplo, la firma Payfec ha producido un sistema de dibujo profesional, denominado DOGS, mientras que Lattice Logic ha desarrollado un paquete de diseño de procesadores denominado Chipsmith

Cortesía de Whitechapel Computer Centre



Identificar las señales

Proseguimos el detallado examen de las patillas individuales del chip convertidor 7135

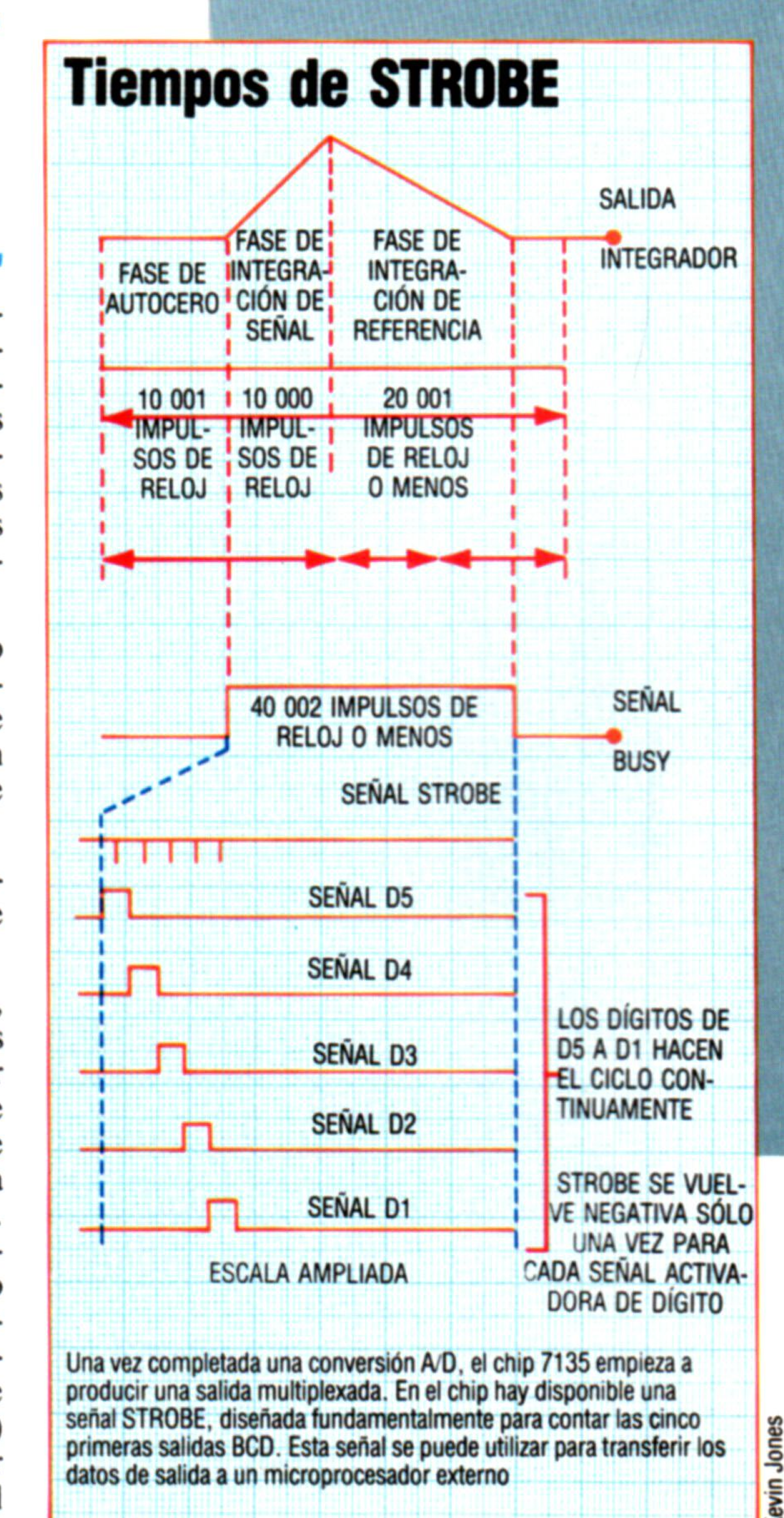
7/8. REF. CAP. (Condensador de referencia): Sorprendentemente, el valor y la calidad de este componente no son muy críticos. Se sugiere un condensador de 1 μF, aunque se podría utilizar uno más pequeño (al precio de una pérdida de eficacia durante los primeros segundos de recuperación tras una sobrecarga). Se han de emplear condensadores de polipropileno, de policarbonato o de poliestireno.

9/10 IN–/IN+: Se dispone de dos entradas así como una masa analógica (patilla 3). Se pueden acomodar entradas diferenciales en cualquier punto entre la escala –4 V y +4,5 V. Estas entradas se pueden dejar flotantes o se puede unir IN– al punto de masa analógica.

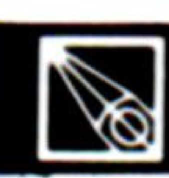
11. V+: Para alimentar el chip se requiere una alimentación de +5 V a una corriente máxima de 3 mA.

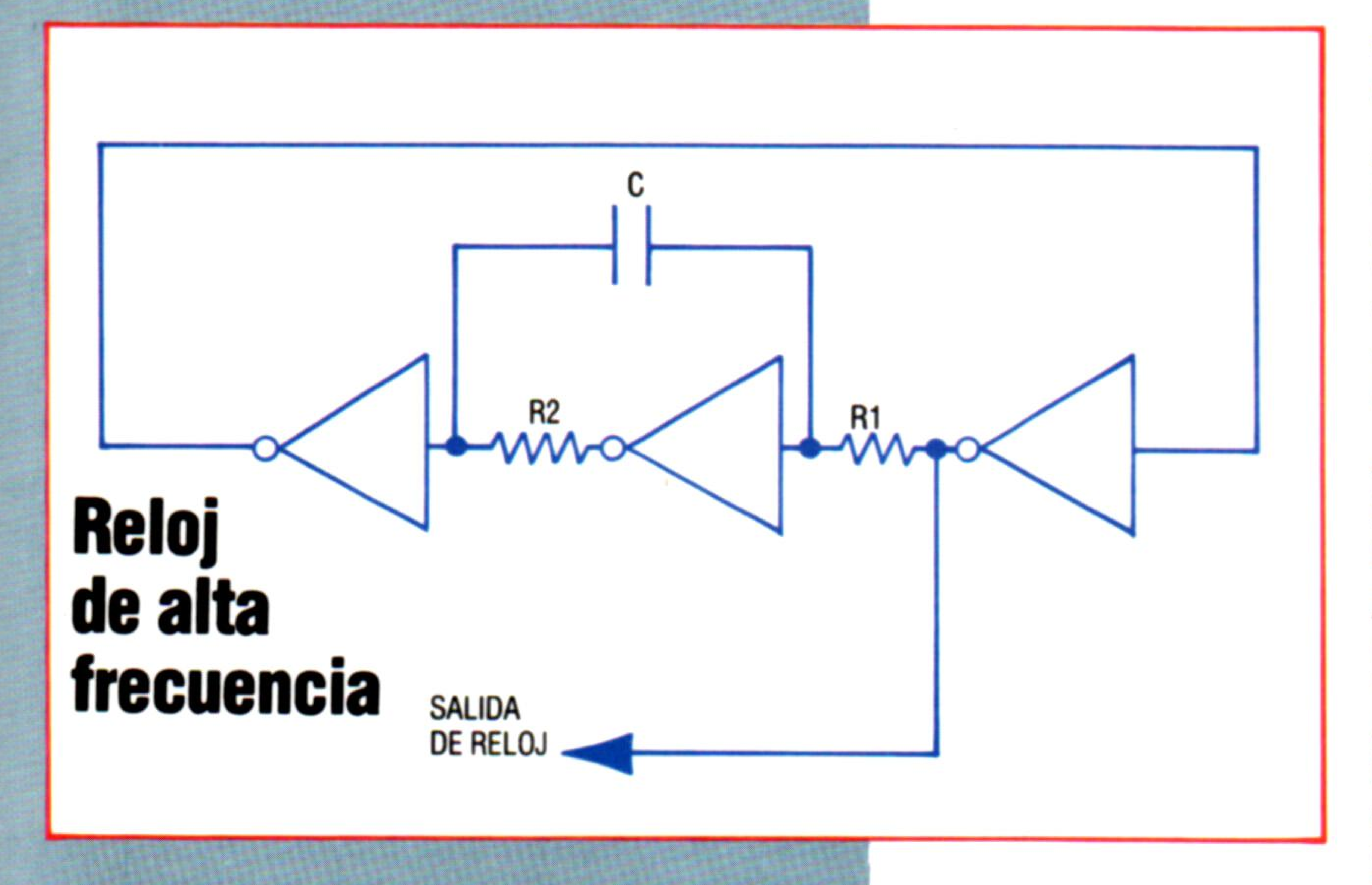
12. D5: Hay cinco salidas de habilitación de dígitos, de D1 a D5. (Las otras salidas están en las patillas 17, 18, 19 y 20.) Se utilizan para habilitar una por una las visualizaciones LED de siete segmentos, de modo que se puedan multiplexar. Cada salida de habilitación de dígitos es una señal activa positiva que permanece alta durante 200 impulsos de reloj. Los cinco dígitos se exploran continuamente, comenzando por D5, el dígito más significativo (MSD) y terminando en D1, el dígito menos significativo (LSD). A medida que cada salida habilitadora de dígito se hace activa, la tensión se eleva desde el nivel casi de masa (aproximadamente 0,25 V) hasta casi la tensión de la línea positiva (alrededor de 4 V). Cuando sucede esto, el transistor al cual está aplicada se hace conductor y permite que circule corriente por el dígito correspondiente (ver esquema del circuito). Las señales activadoras de dígitos sólo *habilitan* cada una de las visualizaciones LED de siete segmentos, al conectar el ánodo común a la línea de + 5 V. Los segmentos individuales de la visualización se encienden mediante las señales BCD (B1, B2, B4 y B8), decodificadas por el chip activador/decodificador de siete segmentos, el 7447. En nuestra serie sobre lógica hemos visto circuitos lógicos que convierten una entrada BCD de cuatro líneas en una salida de siete segmentos que ilumina las barras adecuadas de una visualización de siete segmentos.

13. B1 (LSB): Hay cuatro líneas de salida BCD (*binary coded decimal:* decimal codificado en binario): B1 en la patilla 13, B2 en la patilla 14, B4 en la patilla 15 y B8 (el MSB) en la patilla 16. Entre ellas proporcionan un equivalente codificado en binario del dígito decimal a visualizar en cualquier momento. El código BCD producido será el apropiado para el dígito que esté habilitado entonces. Las señales habilitadoras de dígitos del chip están sincronizadas de manera que el valor BCD para cualquier dígito determinado esté presente en las líneas de salida cuando la correspondiente señal habilitadora sea alta. Los cinco dígitos parecerán estar todos encendidos al mismo tiempo. En realidad están multiplexados, encendiéndose de forma intermitente en secuencia, uno después de otro. Esto da una intensidad de salida luminosa aparentemente mucho menor que si los LED estuvieran iluminados continuamente. Sin embargo, no es práctico encender las cinco visualizaciones al mismo tiempo. Debido a su gran consumo, los LED requerirían casi cinco



Una vez completada una conversión A/D, el chip 7135 empieza a producir una salida multiplexada. En el chip hay disponible una señal STROBE, diseñada fundamentalmente para contar las cinco primeras salidas BCD. Esta señal se puede utilizar para transferir los datos de salida a un microprocesador externo





Reloj de alta frecuencia

Fijación del tiempo
Esta alternativa al chip de reloj 555 utiliza apenas tres inversores TTL, dos resistencias y un condensador. La fórmula para calcular la frecuencia deseada es:

$F = 2C(R1 + R2)$

Puesto que el chip convertidor 7135 tolera velocidades de reloj relativamente altas, este tipo de circuito de reloj se podría utilizar para sincronizar su acción en caso de que se necesitaran mediciones de alta frecuencia y de poca precisión

veces la energía de un circuito con un solo LED encendido cada vez. Aun más importante es el hecho de que se requerirían salidas BCD separadas para cada dígito (20 patillas en lugar de 9, y cada visualización requeriría su propio chip activador de visualización 7447: cinco chips en lugar de uno). Y la única ventaja sería una visualización más brillante. El multiplexado permite una gran reducción de los costos de cableado y de componentes.

21. BUSY: Ésta es una línea importante si desea conectar el DVM en interface con su micro. Es una señal *alta activa,* lo que significa que, cuando es alta, el convertidor A/D está ocupado (*busy*) haciendo cálculos, y que el dígito y las salidas BCD no tienen ningún significado real y se deben ignorar. Se pone alta al principio de la fase de integración de señal, y permanece alta hasta el primer impulso de reloj después del punto paso por cero. Esto significa que cuando la señal BUSY se hace baja, la salida de señal está fijada en un estado estable y puede ser leída válidamente por, pongamos por caso, un microordenador externo. En anteriores capítulos ya hemos analizado este principio.

22. ENTRADA DE RELOJ: Se utiliza una fuente de reloj externo para disparar las operaciones internas del chip 7135. Esta señal de reloj no debe ser superior a 1,2 MHz. Esta frecuencia máxima proporciona una actualización sumamente rápida de la tensión que se esté midiendo, pero una frecuencia tan elevada puede crear muchos problemas relacionados con la selección de componentes y la precisión. En cualquier caso, el chip 555 que utilizaremos no puede generar tales frecuencias: no subirá por encima de 100 KHz.

Dado que un ciclo de medición completo emplea un máximo de 40 002 impulsos de reloj, la frecuencia del reloj determinará cuántas mediciones se harán por segundo. Un reloj de 100 KHz efectuará aproximadamente 2,5 mediciones por segundo. Si usted necesita mediciones más rápidas, son factibles frecuencias de reloj de hasta 1,2 MHz, pero necesitará un circuito oscilador de reloj alternativo y condensadores de autocero y de referencia de muy alta calidad. Es probable que también la precisión se vea afectada. La flexibilidad del chip 7135 se pone de relieve por el hecho de que puede trabajar en una gran gama de frecuencias, incluso de 1 Hz, si bien, en este caso, ¡tendría que esperar 40 002 segundos para una lectura!

23. POLARIDAD: Esta patilla produce una señal que indica simplemente la polaridad de la tensión que se está midiendo. Es positiva para todas las tensiones de entrada positivas, aun para aquellas tan bajas que la visualización indique +00000. Se utiliza para encender el signo "+" en la visualización LED del ½ dígito situado más a la izquierda.

24. MASA DIGITAL: Aquí es muy poco lo que se puede decir que no se haya dicho ya al hablar de la patilla 3, masa analógica.

25. RUN/HOLD (correr/retener): Cuando esta patilla de entrada es alta, el 7135 está en modalidad "funcionamiento libre", dando lecturas de salida cada 40 002 impulsos de reloj o menos. Si se hace baja (ya sea mediante una señal HOLD desde un ordenador, o bien mediante un interruptor del panel del DVM), la lectura actual se mantendrá con carácter indefinido. Tras esta retención, una señal positiva en esta patilla que dure al menos 300 nanosegundos comenzará un nuevo ciclo de medición. Esta patilla sería útil para una interface de ordenador en paralelo lenta, si no se pudieran procesar con rapidez suficiente las cinco salidas BCD.

26. STROBE (impulsos para sincronismo): Hay cinco impulsos STROBE que se hacen negativos (uno para cada dígito válido) en cada ciclo de medición. Están temporizados para producirse en medio de cada impulso de salida de habilitación de dígito (ver gráfico de tiempos). La señal STROBE se proporciona para simplificar la conexión en interface a microprocesadores y se puede utilizar para transferir los datos BCD a registros de retención externos. La señal STROBE se hace verdadera (negativa) sólo durante medio impulso de reloj, y solamente una vez para cada dígito. Los activadores de dígitos y las salidas BCD continúan efectuando ciclos durante el ciclo de medición actual (para permitir la aparición de una visualización continua), pero una interface de ordenador en paralelo tendría que captar los impulsos STROBE cuando se produjeran, ya que no se producirán más hasta el siguiente ciclo de medición (ver gráfico de tiempos).

27. SUPERACIÓN DE MARGEN: Esta señal indica que la tensión de entrada que se está midiendo es demasiado elevada (más de 20 000 puntos). En un diseño más sofisticado, se podría utilizar como parte de un circuito de escala automática (un circuito que estableciera automáticamente la sensibilidad de la entrada sin el uso de conmutadores mecánicos). En un circuito más simple como el nuestro, se podría utilizar para iluminar un LED que indicara que el atenuador de entrada se habría de situar en una sensibilidad inferior (una escala más alta).

28. MARGEN INSUFICIENTE: Esta señal se hace alta cuando la tensión de entrada que se está midiendo es demasiado baja. Al igual que la señal SUPERACIÓN DE MARGEN, está pensada básicamente para emplearse en un circuito de escala automática. Los DVM de escala automática utilizan estas señales para conmutar el atenuador de entrada a una sensibilidad adecuada para la señal que se esté midiendo, pero la escala automática incrementa la complejidad del diseño y en nuestro circuito no utilizaremos esta señal.

Ensayo general

Una vez situado cada personaje en nuestra representación, vamos a manejar sus atributos

Por lo general se puede reconocer un programa de aventuras escrito en BASIC: un apiñamiento de sentencias IF. En un libro muy conocido sobre programación de aventuras en BASIC se incluye un programa de muestra que incorpora no menos de cuatro páginas de sentencias IF y THEN, lo que podría inducir a la conclusión de que este método para tomar decisiones es inevitable cuando se escribe en BASIC.

En las aventuras el problema surge a raíz de la gran cantidad de variables que necesitamos procesar con el fin de decidir la acción a tomar. Por ejemplo, cada uno de los personajes del Dog and Bucket tiene 10 atributos, y nuestro manipulador de personajes necesita poder juzgar y actualizar cada uno de estos atributos cuando manipula a la "persona" en cuestión. Durante la ejecución del manipulador podríamos, por ejemplo, tener una rutina que opere en líneas generales del siguiente modo: "IF el personaje X está en la sala de tertulia AND el personaje Y está en la sala de tertulia AND IF el personaje Y sostiene la bebida del personaje X AND el personaje X sostiene algo, THEN el personaje X le arroja el objeto al personaje Y".

Si programásemos en nuestro juego esta rutina y otras similares, parecería que fuéramos a tener montones de sentencias IF, tanto si lo quisiéramos como si no. Esto no es de desear, ¡sin olvidar el enorme trabajo que tendríamos para entrar el listado! Lo que se necesita, por lo tanto, es una forma ya sea de reducir drásticamente la cantidad de IFs, o bien (lo que sería mejor) arreglárnosla sin ellas directamente, porque, al contrario de lo que se pueda suponer, *sí* existen alternativas viables a la infinidad de sentencias IF. Estas alternativas no sólo son más compactas, sino que también tienden a ser más rápidas en la ejecución, de modo que los beneficios son incuestionables.

Con el objeto de listar las diversas posibilidades de que disponemos, veamos un programa sencillo que nos permita establecer una serie de condiciones, comprobarlas y actuar en consecuencia. Luego examinaremos distintas maneras de codificarlas y decidiremos cuál es el mejor método a adoptar para nuestro manipulador de personajes. Entre en primer lugar el "módulo subrutinas de bajo nivel" (líneas 4000-4140) que dimos en el capítulo anterior (y con los "complementos" para BBC, Commodore o Spectrum), y después digite esto:

```
10 h$="humano  ":a$="animal  ":m$  "macho
   ":f$="hembra ":q$="Eres un ": DIM c(4)
20 GOSUB 4050: REM limpiar la pantalla
30 REM establecer las cuatro variables
40 PRINT q$;h$;:INPUT i$:c(1)=ABS(i$="s" OR
   i$="S")
50 PRINT q$;a$;:INPUT i$:c(2)=ABS(i$="s" OR
   i$="S")
60 PRINT q$;f$;:INPUT i$:c(3)=ABS(i$="s" OR
   i$="S")
70 PRINT q$;m$;:INPUT i$:c(4)=ABS(i$="s" OR
   i$="S")
80 PRINT
```

La línea 10 de este módulo primero establece las variables en serie utilizadas en las líneas 40-70, y luego DIMensiona una matriz para cuatro elementos. Esta matriz se emplea luego para almacenar cuatro valores condicionales, según las respuestas dadas a las preguntas formuladas. Para la mayoría de los micros, los valores condicionales son 0 para falso y −1 para verdadero (si bien el Spectrum utiliza 1). La condición negativa explica por qué hemos incluido ABS en cada asignación.

Al ejecutar (RUN) este módulo, se creará la matriz c(4), lista para la comprobación ulterior. Sin embargo, es en este punto donde surge el problema, ya que con cuatro condiciones diferentes hay 16 posibles combinaciones distintas. Si usted sólo quisiera probar un caso (si, p. ej., el usuario hubiera entrado "S" a la primera pregunta y "N" a todas las otras) fácilmente podría utilizar una construcción IF...THEN. No obstante, si quisiera probar, pongamos por caso, 15 de las 16 combinaciones posibles, podría acabar con 15 líneas así:

```
 90 REM Imprimir mensajes
100 IF c(1)=1 AND c(2)=1 AND c(3)=1 AND
    c(4)=1 THEN PRINT "Eres un hombre lobo
    mixto!"
110 IF c(1)=1 AND c(2)=1 AND c(3)=1 AND
    c(4)=0 THEN PRINT "Eres un hombre lobo
    hembra!"
```

...y así sucesivamente. Esto, obviamente, resulta más bien largo, y hay otros métodos que podemos considerar.

El primero es usar la construcción ON...GOTO (si su micro la tiene; la mayoría disponen de ella), pero ésta también tiene sus limitaciones, porque requiere una secuencia de valores y tiende a ser bastante inflexible. Sin embargo, por regla general, es una alternativa muy poco utilizada para las sentencias IF y en algunas máquinas (como el Amstrad, p. ej.), también ofrece la ventaja de una ejecución más rápida.

El segundo método a investigar es de hecho el que adoptaremos para nuestro manipulador de personajes. Supone replantearse el problema en cierta medida y nos proporcionará un potente medio para clasificar distintas combinaciones de condiciones a medida que vayan surgiendo en el Dog and Bucket. La respuesta reside en la implementación de "árboles de decisión" para nuestros personajes.

Retrocedamos a las cuatro condiciones que establecimos en nuestro programa y, en vez de seleccionar las condiciones que deseamos responder mediante el empleo de sentencias IF, veamos cómo podríamos representar el proceso de decisión como una estructura arborescente. El diagrama muestra el posible aspecto de este árbol.

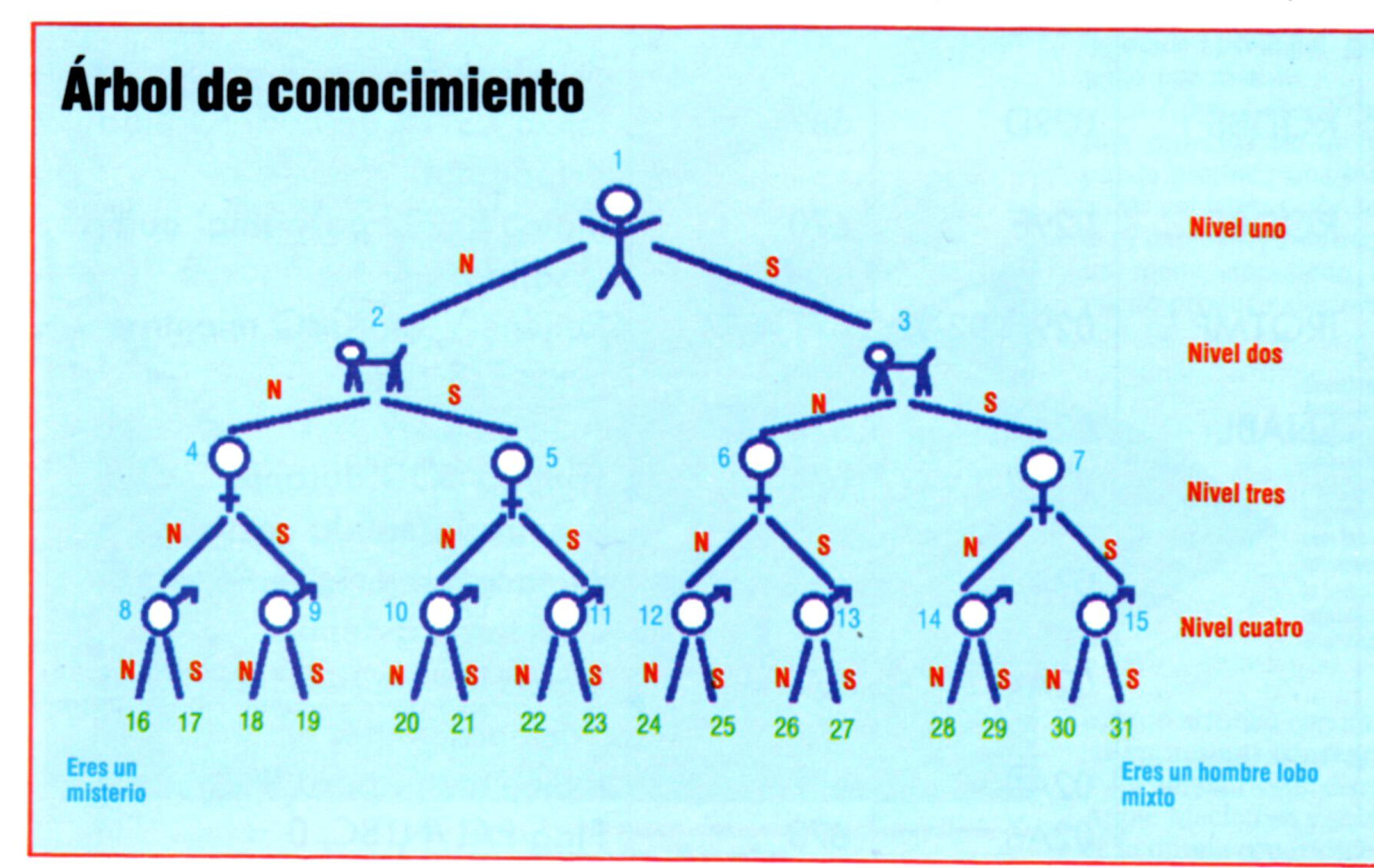
Puede ver que la primera condición a comprobar, en el nudo 1, es "¿humano?" Si la respuesta es sí (en otras palabras, si la variable de matriz

c(1)=1), entonces bifurcamos al nudo 3; de ser falsa, bifurcamos al nudo 2. Este proceso continúa hasta que llegamos a la parte inferior del árbol, uno de los "nudos terminales", numerados del 16 al 31. Cada uno de estos nudos terminales podría tener un mensaje o una rutina. Se han incluido ejemplos para el nudo 16, al que se llega cuando el usuario ha entrado una respuesta negativa a las cuatro preguntas, y para el nudo 31.

Antes de intentar calcular cómo implementar tal estructura dentro de nuestro programa, hay un par de otros puntos que debemos resaltar. En primer lugar, usted puede ver que el árbol está dividido en cuatro "niveles" y que en cada nivel todas las condiciones a probar son las mismas. El nivel 1 comprueba la condición "humano", el nivel 2 la condición "animal", el nivel 3 la condición "hembra" y el nivel 4 la condición "macho". Por el diseño, los números de nivel coinciden con los elementos de la matriz c(4), de modo que c(1) retiene la condición "humano", y así sucesivamente.

```
140 PRINT "Pulsa una tecla para continuar..."
150 GOSUB 4130:REM tomar un personaje
160 GOTO 20
170 REM datos para mensajes
180 DATA "misterio", "mineral macho?!", "mineral hembra?!", "ridículo despilfarrador de tiempo", "animal neutro", "animal macho", "animal hembra", "serpiente? Pueden ser bisexuales, ¿sabes?", "humano, sexo desconocido", "hombre", "mujer", "niño mixto", "hombre lobo neutro", "hombre lobo macho"
190 DATA "hombre lobo hembra", "hombre lobo mixto"
```

Veamos ahora cómo funciona este listado. Deseamos empezar en el nudo 1 y llevar el registro del lugar del árbol donde nos hallamos, de modo que lo primero que hemos de hacer es inicializar una variable n para representar el número de nudo actual. Luego la línea 110 hace todo el trabajo de ir desplazándonos por el árbol, nivel a nivel, acabando en uno de los nudos terminales. Esto lo hace utilizando la variable k para representar el nivel aplicando la fórmula descrita arriba, número de nudo = (2* número de nudo) + c (número de nivel), para decidir hacia qué nudo bifurcarse para ir descendiendo por el árbol.



Un "árbol de decisión" puede ser una forma muy útil para determinar el flujo del programa cuando es necesario tomar en consideración gran número de condiciones. El árbol que vemos es uno binario simple; en BASIC se pueden programar formas más complejas, si bien esta clase de estructura de datos idealmente es más adecuada para lenguajes como el LISP, el LOGO o el PASCAL. Aquí el usuario entra valores para cuatro condiciones diferentes, que pueden luego ser clasificadas por el árbol para llegar a una de 16 conclusiones diferentes

Caroline Clayton

El siguiente punto a observar es la relación entre los distintos números de nudo. Cada nudo elegido se bifurca a dos nudos inferiores, y los números de éstos se podrían determinar mediante:

número del nudo inferior = (número de nudo elegido* 2)+valor de la condición

donde el valor de la condición, retenido en la matriz c, es 0 o bien 1.

Introduzca el tercer listado y ejecute el programa, tras lo cual será saludado con un mensaje adecuado, según cuáles hayan sido sus respuestas a las cuatro preguntas.

```
90 REM ordenado del árbol
100 n=1: REM empezar por el nudo 1
110 FOR k=1 TO 4: n=(2*n)+c(k): NEXT k
120 RESTORE:REM establecer señalador de datos en comienzo mensajes
130 FOR x=1 TO n-15: READ z$:NEXT x: PRINT "Eres un ";z$: PRINT
```

Una vez que hemos recorrido los cuatro niveles y llegado a uno de los nudos del 16 al 31, es muy sencillo restarle 15 al número de nudo terminal para obtener un número entre 1 y 16 y luego, en la línea 130, leer el almacenamiento DATA y seleccionar un mensaje adecuado para imprimir. Las líneas 140-160 simplemente esperan a que usted pulse una tecla antes de repetir el proceso.

Existen otras formas en las que podríamos resolver los problemas de nuestro programa "humano/animal". No obstante, la utilidad de las estructuras arborescentes se hace evidente cuando deseamos aplicarle a nuestro procesamiento de condiciones unas reglas bastante más irregulares.

Datos básicos (VIII)

Proseguimos nuestro detallado análisis del mapa de memoria del Commodore 64

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
BAUDOF	0299–029A	665–666	Velocidad baudios de RS232: tiempo completo del bit (µs)
RIDBE	029B	667	Índice RS232 para final buffer entrada
RIDBS	029C	668	Inicio RS232 de buffer entrada (página)
RODBS	029D	669	Inicio RS232 de buffer salida (página)
RODBE	029E	670	Índice RS232 para final buffer salida
IRQTMP	029F–02A0	671–672	Retiene Vector IRQ mientras las E/S cinta
ENABL	02A1	673	Activa RS232
	02A2	674	Sentido TOD durante entrada/salida cassette
	02A3	675	Almacenamiento temp. para lectura cassette
	02A4	676	Indicador temp. D1IRQ para lectura cassette
	02A5	677	Índice temp. para línea
	02A6	678	Flag PAL/NTSC, 0 = NTSC, 1 = PAL
	02A7–02FF	679–767	No utilizado
IERROR	0300–0301	768–769	Vector: mensaje error impresión en BASIC
IMAIN	0302–0303	770–771	Vector: inicio en caliente BASIC
ICRNCH	0304–0305	772–773	Vector: texto *tokenizado* BASIC
IQPLOP	0306–0307	774–775	Vector: listado texto BASIC
IGONE	0308–0309	776–777	Vector: despacho caracteres BASIC
IEVAL	030A–030B	778–779	Vector: evaluación *token* BASIC



Calidad suprema



Chris Stevens

La impresora Apple LaserWriter, gracias a la tecnología óptica y del láser, obtiene una calidad de impresión sin precedentes en ningún micro

La mayoría de las impresoras para ordenadores son de tipo matricial o bien de rueda margarita. Si bien éstas son adecuadas para la mayoría de las aplicaciones, cada una posee sus propios inconvenientes. Las impresoras matriciales o de matriz de puntos, aunque veloces y capaces de producir una enorme gama de caracteres diferentes, producen un tipo de letra que raramente resiste un cuidadoso examen. La impresora matricial promedio, que en su cabeza posee alrededor de 8 × 10 agujas, produce un perfil bastante irregular. Por su parte, las impresoras de rueda margarita pueden proporcionar una impresión de gran calidad, pero carecen de la velocidad y la versatilidad de las de tipo matricial. Por ejemplo, una de rueda margarita sólo posee un tipo (a menos que el usuario desee cambiar la rueda), con una velocidad máxima de alrededor de 80 caracteres por segundo, mientras que una de tipo matricial puede producir textos a velocidades de 2 a 300 caracteres por segundo, aunque a esa última velocidad es probable que la calidad de la letra se vea aún más disminuida. Las impresoras láser son capaces de producir páginas impresas de calidad a velocidades asombrosas. Por ejemplo, las impresoras láser IBM 3800 Modelo 3, para usar con sistemas IBM de unidad principal, puede producir 215 páginas de texto por minuto.

La Apple LaserWriter, que es una versión a escala reducida de la impresora láser estilo IBM, puede producir una salida impresa de gran calidad a 300 caracteres por segundo. Además, al usuario le es dable elegir entre numerosos tipos distintos de un menú que tiene a su disposición. También puede producir diagramas, formas y otros tipos de trabajo artístico con un estándar muy superior al de las impresoras convencionales.

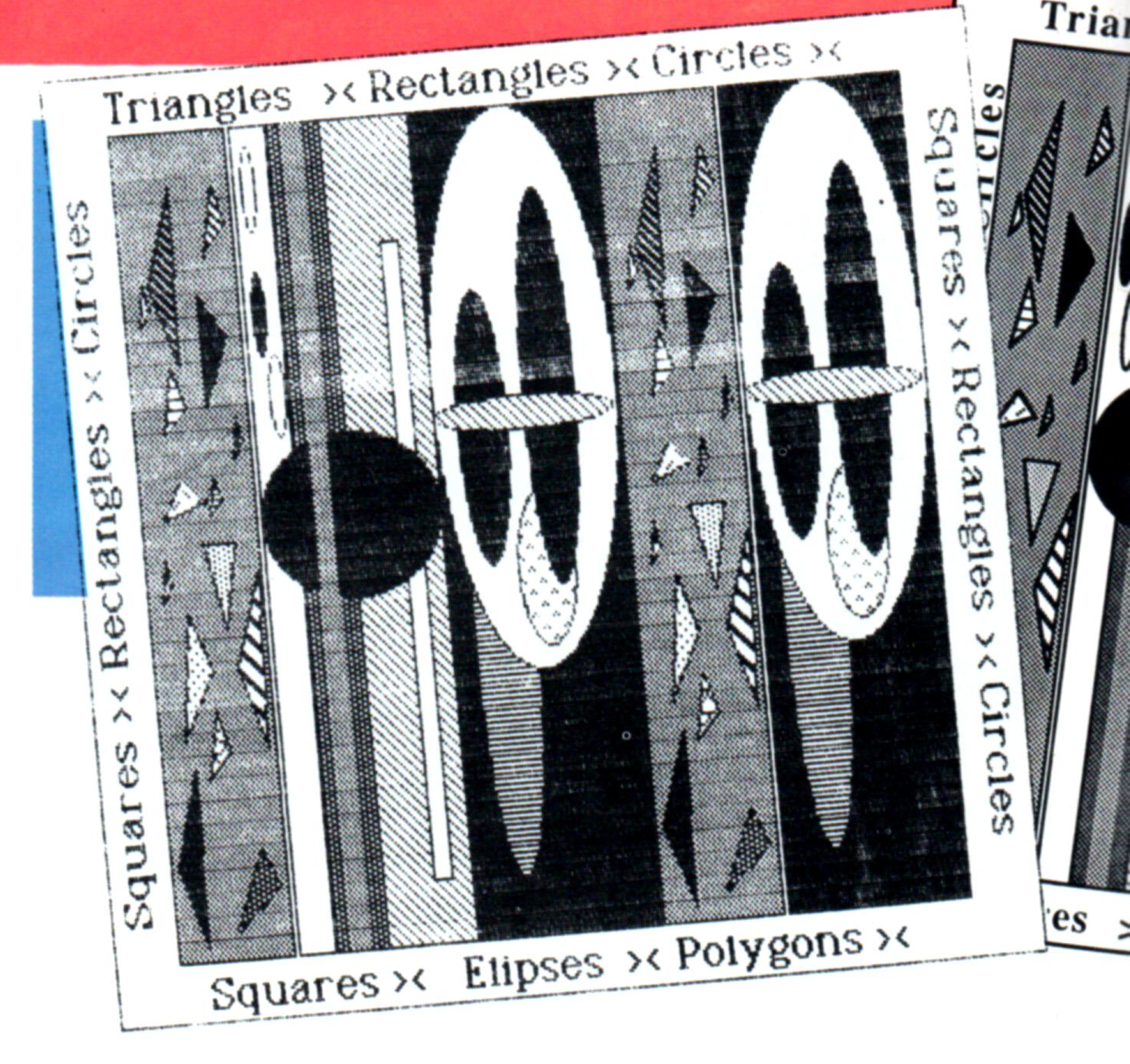
Excelente impresión
Aunque la LaserWriter está pensada para su utilización con micros, es totalmente distinta de las impresoras matriciales y de rueda margarita que se usan con la mayoría de ordenadores. Sería más acertado compararla con las veloces impresoras láser para miniordenadores y superordenadores. Si bien la LaserWriter es más lenta que sus equivalentes para ordenadores centrales, su avanzada tecnología es similar

La unidad está diseñada para utilizarse con el Apple Macintosh y se conecta al ordenador a través de la misma conexión en serie de alta velocidad utilizada para otros tipos de impresoras.

La LaserWriter trabaja de una forma radicalmente diferente a la de las unidades convencionales. Mientras que las impresoras matriciales y de rueda margarita son esencialmente desarrollos del sistema de máquina de escribir, con una cabeza móvil que escribe sobre un rollo de papel, la LaserWriter se aproxima más a una fotocopiadora.

La impresora mide 290 por 420 por 475 mm y alberga dos tambores, el primero de los cuales contiene el toner fotocopiador, mientras que el segundo contiene el equipo láser y el cilindro rotativo que se utiliza para imprimir el dibujo o el texto sobre el papel.

El mecanismo de impresión, diseñado por Canon, trabaja tomando la página a imprimir y convirtiéndola en una imagen de bits. La imagen se transfiere luego al cilindro, que está cubierto por millones de puntos eléctricamente sensibles. Antes de que se produzca la transferencia, los puntos se ionizan de modo que retengan una carga eléctrica



positiva. Para transferir la imagen, el láser realiza un barrido sobre el cilindro, "escribiendo" la página en su superficie, neutralizando la carga positiva de los puntos sobre los que incide. El toner, que también está cargado positivamente, se utiliza, entonces, para cubrir la superficie del cilindro. El toner se "pegará" en los puntos neutralizados, pero evitará los demás, que, por tener la misma carga positiva, lo repelerán.

Para transferir el toner al papel, la LaserWriter aplica una carga negativa al papel a medida que éste pasa por la máquina. Cuando el papel pasa por el tambor, el toner cargado positivamente es atraído hacia el papel cargado negativamente y se adhiere a él. Para fijar el toner sobre la página de forma permanente, el papel pasa luego a través de un par de rodillos calentados a una temperatura de 200 °C. El toner se funde sobre el papel, que sale, entonces, de la máquina. Aplicando este sistena la LaserWriter puede producir hasta ocho páginas completas por minuto. Al funcionar, la LaserWriter genera una gran cantidad de calor, si bien éste no lo produce la máquina propiamente dicha, sino los rodillos.

Después de producir aproximadamente 3 000 páginas, algunos de los componentes del sistema láser comienzan a deteriorarse. Estos componentes están fijados al tambor del toner, y cuando el toner se acaba, no existe medio alguno de recargar el tambor, de modo que hay que sustituirlo por completo, junto con los componentes desgastados. Si bien este sistema puede parecer más bien caro, Apple señala que si se tiene en cuenta el precio de cada copia, la LaserWriter está a la altura de la mayoría de las fotocopiadoras corrientes.

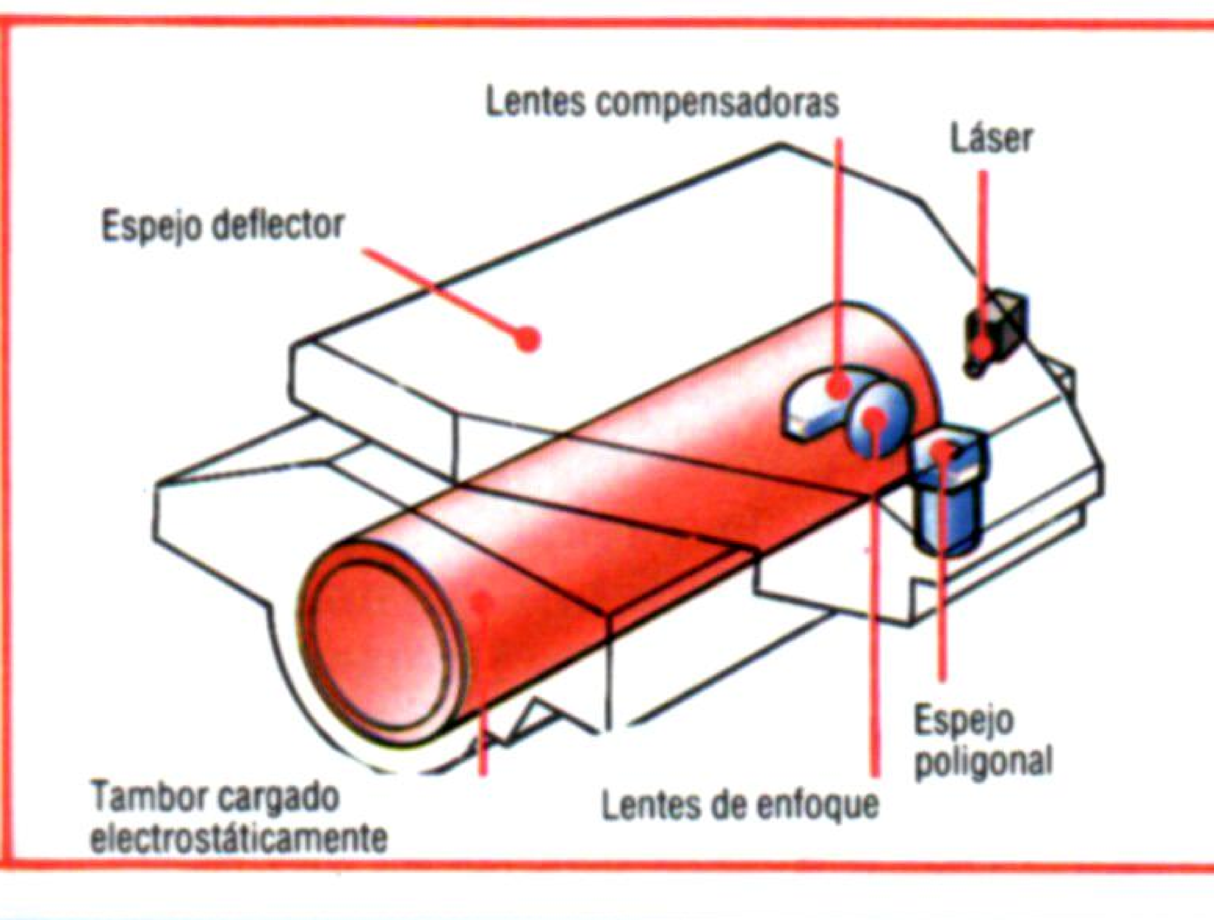
La calidad de impresión producida es similar a la de la fotocomposición (como la utilizada para producir esta página). Esto se consigue en virtud de la cantidad de puntos que Canon ha logrado apiñar en la superficie del cilindro: 300 puntos por pulgada. Para una página de tamaño A4, esto significa que para su producción se utilizan casi siete millones de puntos.

Este sistema de impresión láser, aunque rápido y eficaz, no es de ningún modo exclusivo. Lo que diferencia a la LaserWriter de dispositivos similares existentes en el mercado, es la forma en que Apple ha incorporado en la mecánica su propia tecnología, convirtiéndola en la primera impresora "inteligente". Ya hemos visto que la LaserWriter coloca en su memoria una imagen de la página antes de iniciar la impresión. Teniendo en cuenta la altísima resolución de la impresión resultante, es necesario disponer de una considerable cantidad de memoria. La LaserWriter incorpora un ordenador en la placa, destinado exclusivamente a producir la página impresa.

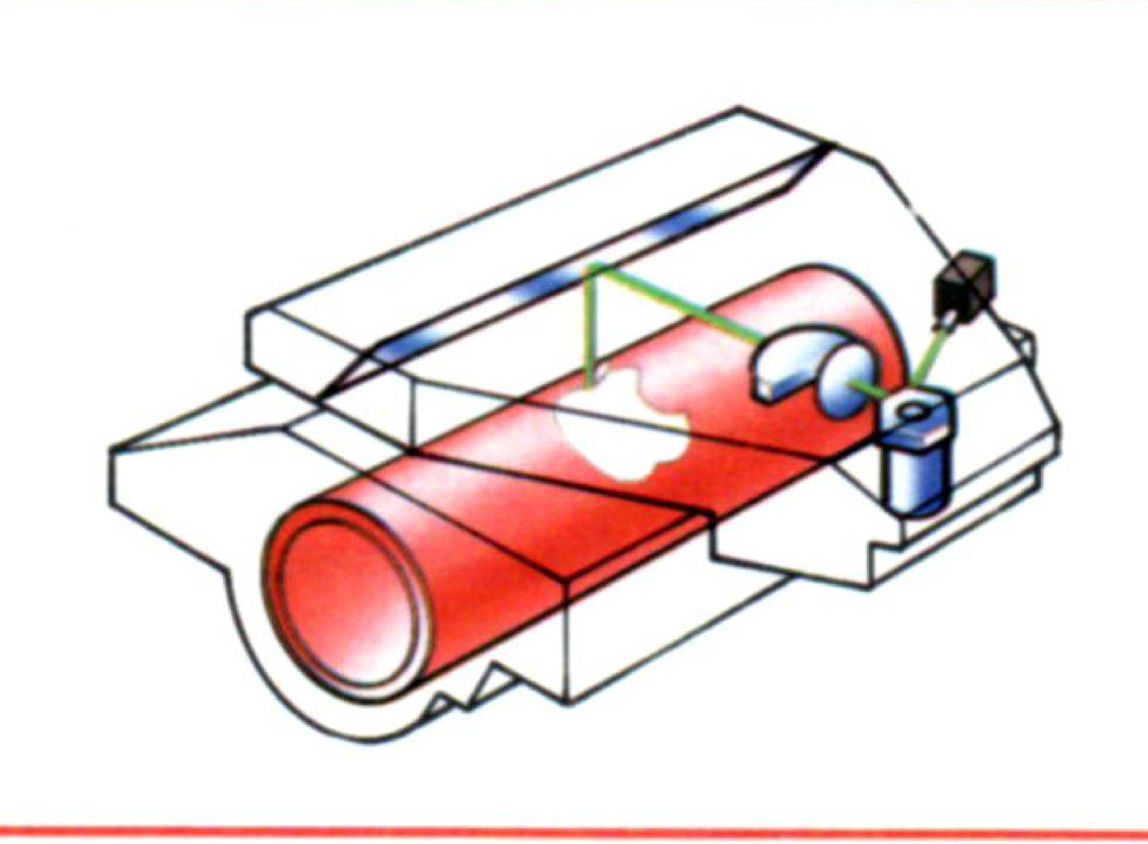
Basada en el procesador Motorola 68000 y trabajando a 12 MHz, el mismo que utiliza el propio Macintosh, la LaserWriter posee 1,5 Mbytes de RAM y 500 Kbytes de ROM. Las ROM albergan juegos de caracteres completos para hasta 13 tipos de letra distintos. No obstante, incluso 2 Mbytes no pueden contener las imágenes completas de mapa de bits para todos ellos y todos los programas de aplicaciones que requiere la LaserWriter. Por lo tanto, con el objeto de ahorrar memoria, la ROM de la impresora se limita a retener la silueta de cada carácter con una rutina especial que "rellena" los caracteres cuando se imprimen. Este sistema no sólo ahorra espacio sino que, al utilizar la misma

Procedimiento perfeccionado
Esta serie de ilustraciones muestran cómo la LaserWriter fija una imagen en el papel. Si bien la tecnología empleada es sumamente sofisticada, los principios son bastante simples y se basan en las técnicas de xerografía que se han venido utilizando desde hace ya muchos años

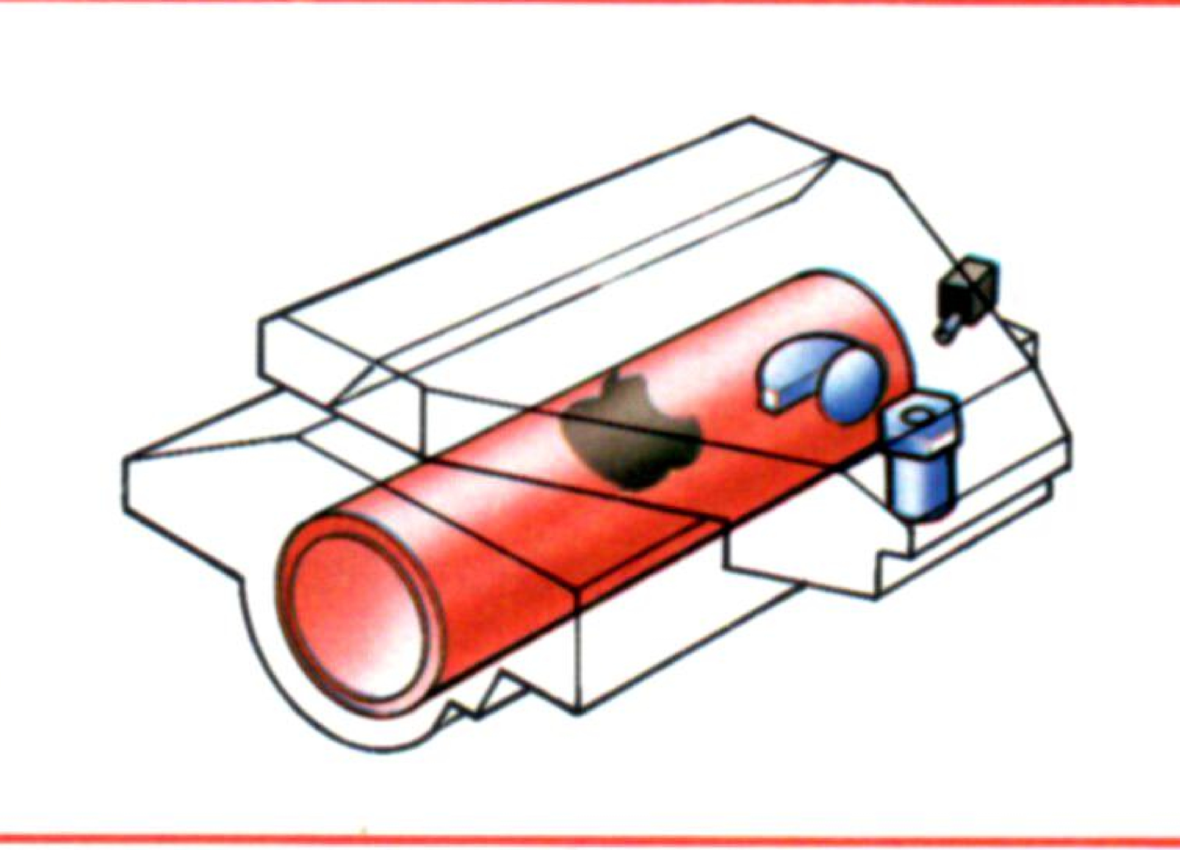
Al tambor se le aplica una carga electrostática positiva



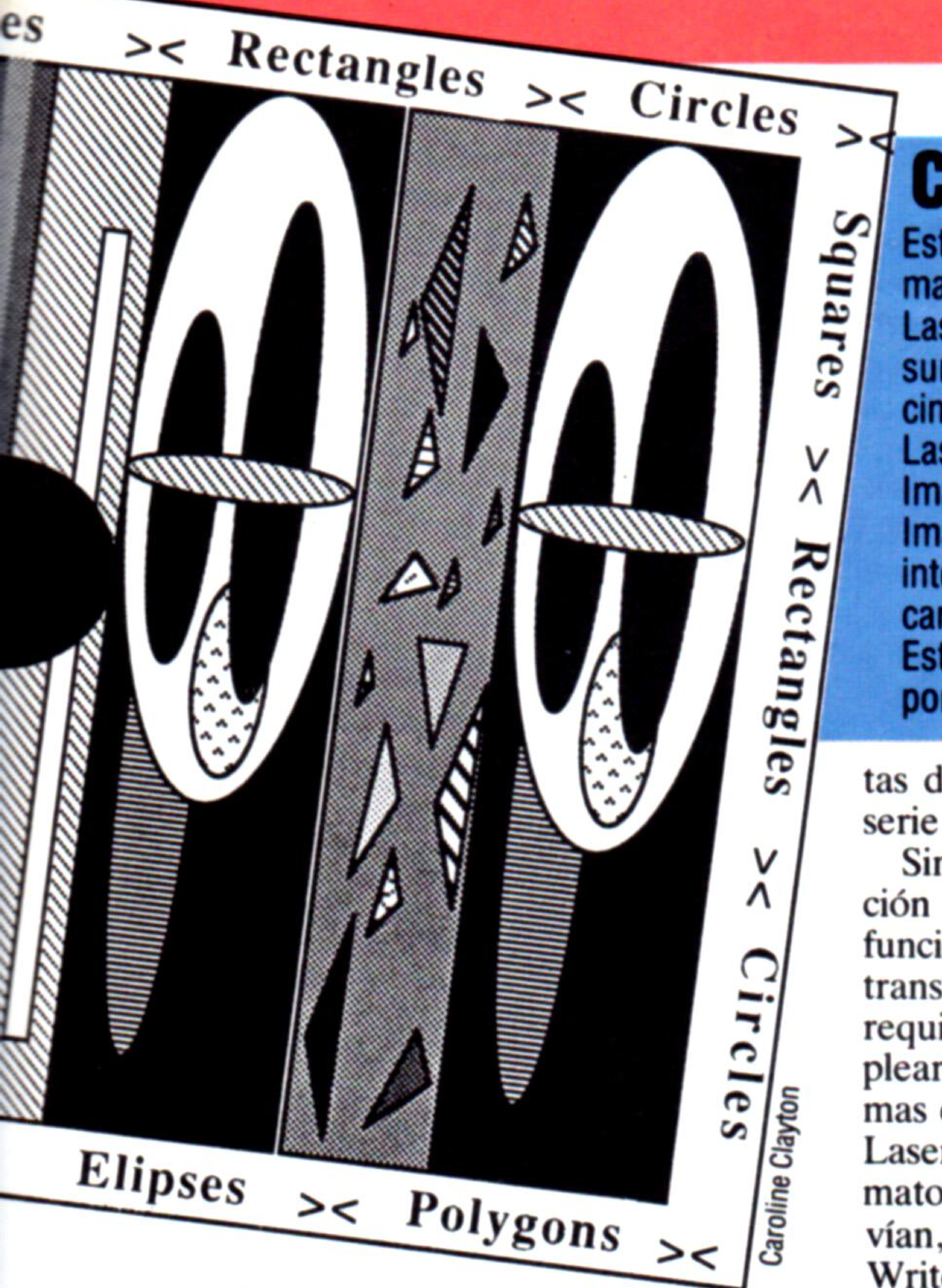
Luego se dispara el rayo láser y se desvía sobre el tambor, donde dibuja la imagen mediante la neutralización de los iones positivos



A medida que la superficie del tambor va pasando por el receptáculo del toner, éste, cargado positivamente, es atraído hacia las zonas "neutralizadas" y adherido a ellas



Kevin Jones



Caroline Clayton

Comparación ilustrativa

Este dibujo se ha reproducido tanto en la impresora matricial Macintosh Imagewriter como en la LaserWriter. Aparte de los problemas que pueden surgir por la variación de la densidad de tinta en la cinta de la impresora matricial, la resolución de la LaserWriter es muchísimo mayor que la de Imagewriter. Un examen atento del dibujo de la Imagewriter revela la aparición de líneas no intencionales. Es allí donde se superponen los caracteres formados por la cabeza de impresión. Este problema no se presenta en la LaserWriter porque la imagen se fija en un tambor único

"plantilla", la silueta se puede expandir para permiir cualquier tamaño de tipo que se requiera.

También retenido en ROM hay un lenguaje es-pecial desarrollado para usar con impresoras, que e denomina PostScript. Se trata de un lenguaje de programación interactivo que permite que el programador tome las siluetas de los tipos retenidos en ROM (y otras facilidades tales como líneas) y manipularlas para producir muchos estilos y diseños. Las instrucciones de que consta el lenguaje se pueden utilizar para crear rutinas que produzcan círculos, etc., que, a su vez, se pueden enlazar entre sí para producir diagramas complejos. En operación, el lenguaje se asemeja mucho al FORTH en su intenso uso de una pila para pasar datos desde y hacia os procedimientos, y en su capacidad para definir otros procedimientos. PostScript trabaja de una forma muy distinta a la de otros lenguajes de programación de gráficos tales como el LOGO. En vez de decirle a la impresora que dibuje una serie de líneas en términos absolutos, PostScript utiliza una serie de puntos (como los que componen las siluetas de tipos) que se espaciarán de acuerdo a una serie de proporciones.

Sin embargo, el PostScript es más que una opción adicional deseable: es fundamental para el funcionamiento eficaz de la LaserWriter. Para transferir el Mbyte completo de información que se requiere para una página entera, el Macintosh emplearía alrededor de 35 segundos. Pero los programas del disco de sistemas que se suministra con la LaserWriter convierten esta información en el formato Quickdraw del Macintosh. Los datos se envían, entonces, a la línea en serie de la LaserWriter, donde se convierten en instrucciones PostScript. Éstas, a su vez, son interpretadas por el ordenador del LaserWriter para reproducir la página como una imagen por mapa de bits dentro de la memoria del ordenador. El láser usa luego este mapa de bits para "dibujar" la página en el tambor electrosensible. Al reducir así la información, son suficientes 8 K de información, aminorando la velocidad de transmisión y el procesamiento.

La LaserWriter es, sin duda alguna, un paso trascendental en la tecnología de impresoras, no sólo por el avanzado mecanismo láser utilizado para producir una impresión rápida y de gran calidad, sino también por el ordenador proporcionado en la placa y el empleo del lenguaje de programación PostScript. Por supuesto, su precio la coloca muy lejos del alcance de la mayoría de usuarios del Macintosh. No obstante, es tradicional que la tecnología avanzada vaya introduciéndose en las máquinas de precio más económico y, dentro de pocos años, es razonable esperar que gran parte de la tecnología punta utilizada en la LaserWriter esté disponible en equipos más económicos.

APPLE LASERWRITER

DIMENSIONES
290 × 420 × 475 mm

INTERFACES
Interfaces en serie RS422 y RS232C

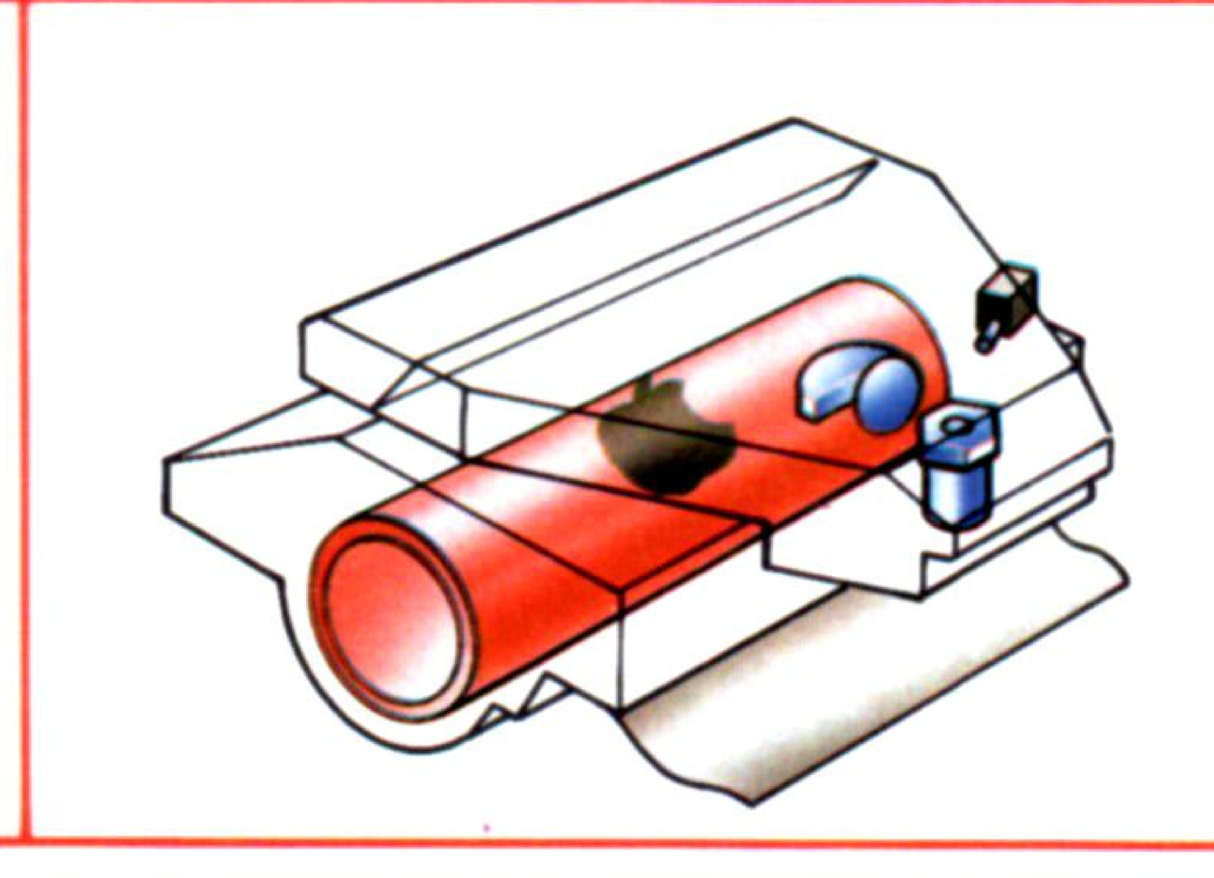
RESOLUCION DE IMPRESION
300 puntos por pulgada

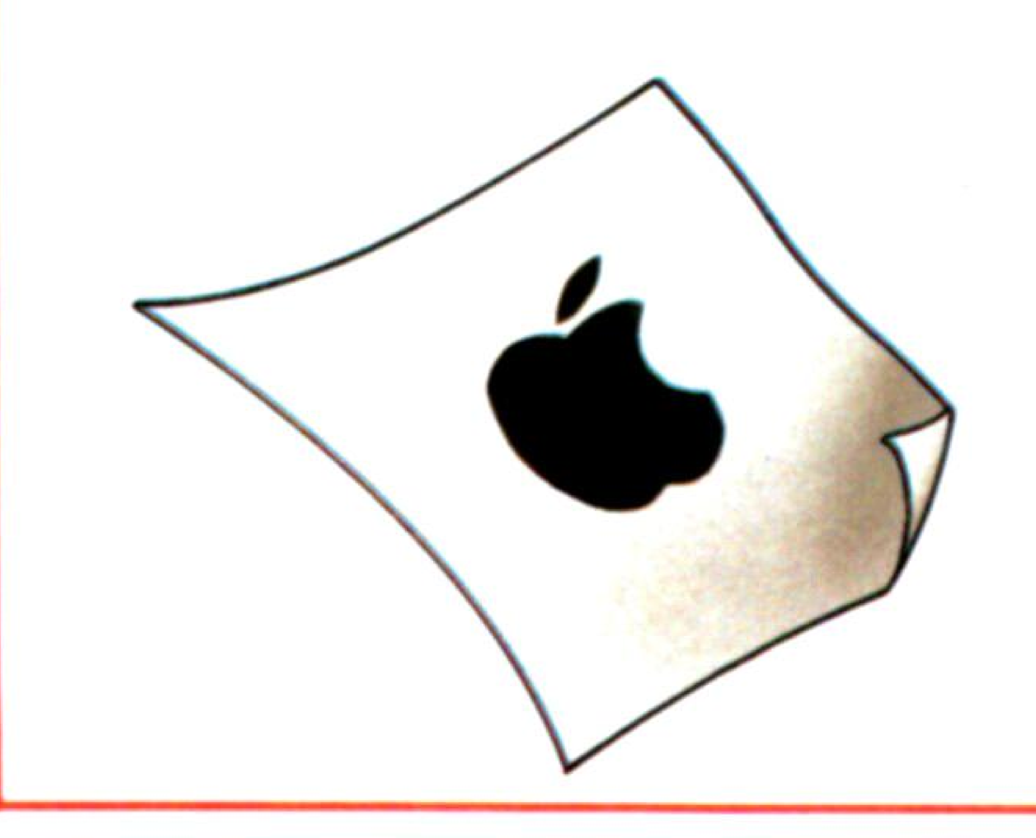
CPU
Procesador Motorola 68000 trabajando a 12 MHz

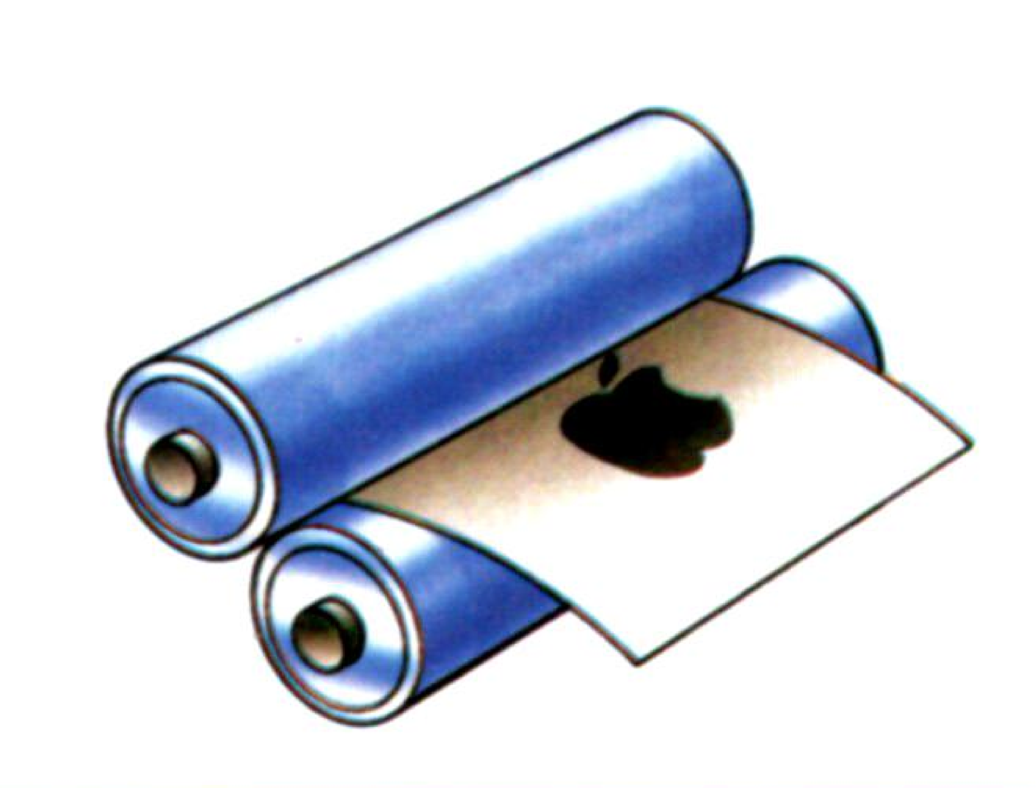
MEMORIA
1,5 Mbytes de RAM y 500 Kbytes de ROM

VENTAJAS
La LaserWriter produce letras y dibujos de calidad profesional a una velocidad muy superior a la mayoría de las otras impresoras diseñadas para micros

DESVENTAJAS
El tamaño máximo del papel es un poco mayor que el estándar A4, lo que limita severamente la posibilidad de producir trabajos artísticos y diseños. Además, por su precio, la máquina no está al alcance de todos los usuarios de micros


Luego se aplica una carga negativa al tambor del papel...


...que atrae al toner con carga positiva y forma una imagen en la página


Por último, el papel que retiene el toner pasa entre dos rodillos calentados que "fijan" el toner en la página

Estrategias oblicuas

Completamos los programas para los micros Amstrad, Commodore 64 y Spectrum añadiendo las rutinas finales de evaluación de movimientos

Uno de los motivos por los que ha resultado tan difícil programar el juego del *go*, es que es casi imposible simular el funcionamiento de la mente humana utilizando la tecnología actual. Para determinar los movimientos, el jugador humano puede analizar intuitivamente los formatos de grupos y líneas formados por las fichas. Puesto que los micros de hoy en día sólo pueden funcionar secuencialmente (es decir, un paso por vez), el análisis de las formas de los grupos plantea un problema. No obstante, podemos incluir rutinas que analicen las conexiones entre grupos adyacentes, permitiendo que el ordenador conecte y defienda dos de sus grupos, o ataque posibles conexiones entre grupos rivales. En este capítulo ofrecemos rutinas que llevan a cabo estas funciones en los micros Amstrad, Commodore 64 y Spectrum.

Módulo 5

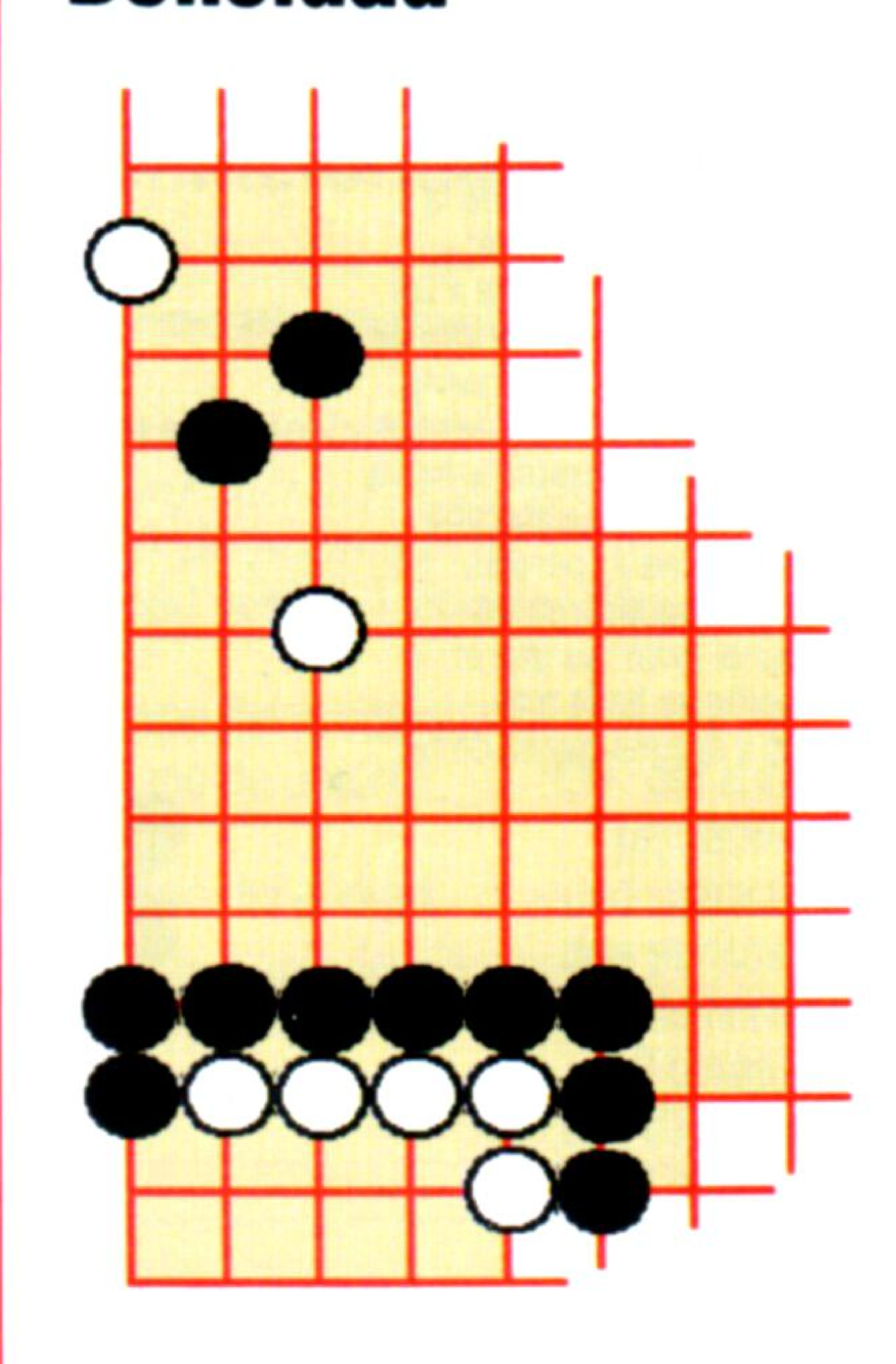
Commodore 64:

```
330 GOSUB 790
780 :
790 REM RUTINA LEER FORMATOS
810 PAT=TABLERO+556
830 FOR L=0 TO 71
840 READ P%:POKE PAT+L,(256+P%) AND 255
850 NEXT
860 DATA 32,17,16,2,-15,1
870 DATA -32,-17,-16,-2,15,-1
880 DATA 33,16,17,31,16,15
890 DATA -33,-16,-17,-31,-16,-15
900 DATA -14,1,-15,18,1,17
910 DATA 14,-1,15,-18,-1,-17
920 DATA 48,33,32,48,17,16
930 DATA -48,-33,-32,-48,-17,-16
940 DATA 3,-14,2,3,-15,1
950 DATA -3,14,-2,-3,15,-1
960 DATA 64,33,32,4,-14,2
960 DATA -64,-33,-32,-4,14,-2
980 RETURN
990 :
1000 REM ************************************
2580 IF POSIT%=0 THEN GOSUB 2900:T$="DEF"
2590 IF POSIT%=0 THEN GOSUB 3070:T$="ATT"
2600 IF POSIT%=0 THEN GOSUB 3230:T$="SAT"
2610 IF POSIT%=0 THEN GOSUB 3380:T$="SCN"
2890 :
2900 REM CONEXION DE DEFENSA
2910 HI=-9999
2920 BV%=NEGRAS%:GOSUB 4420
2930 FOR A=1 TO 255:DCS%=PEEK(TABLERO+A):IF
     DCS%<>NEGRAS% GOTO 3010
2940 FOR P=PAT TO PAT+70 STEP 3
2950 B=PEEK(TABLERO+ ((A+PEEK(P))AND 255)):C
     =PEEK(TABLERO+ ((A+PEEK (P+1))AND 255))
2960 IF B<>NEGRAS% OR C<>BLANCAS% GOTO 3000
2970 D=(A+PEEK(P+2)) AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+D)
2980 IF(D AND 240)=0 OR (D AND 15)=0 OR CSR<=HI
     GOTO 3000
2990 LP%=D:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=D
3000 NEXT
3010 NEXT
3020 BV%=0:GOSUB 4420
3030 RETURN
3040 :
3050 REM ************************************
3060 :
3070 REM RUTINA CONEXION DE ATAQUE
3080 HI=-9999
3090 BV%=BLANCAS%:GOSUB 4420
3100 FOR A=1 TO 255:ACS%=PEEK(TABLERO+A): IF
     ACS%<>BLANCAS% GOTO 3180
3110 FOR P=PAT TO PAT+70 STEP 3
3120 B=PEEK(TABLERO+ ((A+PEEK(P)) AND 255)):C
     =PEEK(TABLERO+ ((A+PEEK (P+1))AND 255))
3130 IF B <>BLANCAS% OR C<>NEGRAS% GOTO 3170
3140 D=(A+PEEK(P+2))AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+D)
3150 IF (D AND 240)=0 OR (D AND 15)=0 OR SCR< = HI
     GOTO 3170
3160 LP%=D:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%> 2 THEN HI=SCR: POSIT%=D
3170 NEXT
3180 NEXT:BV%=GOSUB 4420
3190 RETURN
3200 :
3210 REM ************************************
3220 :
3230 REM RUTINA INICIAR ATAQUE
3240 HI=-9999
3250 FOR A=1 TO 255:SAS%=PEEK(TABLERO+A): IF
     SAS%<>BLANCAS% GOTO 3330
3260 FOR P=PAT TO PAT+70 STEP 3
3270 B=PEEK(TABLERO+((A+PEEK(P))AND 255)):D
     =PEEK(TABLERO+((A+PEEK(P+2))AND 255))
3280 IF B<>BLANCAS% OR D<>0 GOTO 3320
3290 C=(A+PEEK(P+1))AND
     255:SCR=RND(0)+PEEK(ESTIMACIONES+C)
3300 IF (C AND 240)= OR (C AND 15)=0 OR SCR < = HI
     GOTO 3320
3310 LP%=C:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=C
3320 NEXT
3330 NEXT
3340 RETURN
3350 :
3360 REM ************************************
3370 :
3380 REM RUTINA INICIO CONEXION
3390 HI=-9999
3400 FOR A=1 TO 255:SCS%=PEEK(TABLERO+A): IF
     SCS%<>NEGRAS% GOTO 3470
3410 FOR P=PAT TO PAT+70 STEP 3
3420 C=PEEK(TABLERO+((A+PEEK(P+2))AND 255)): IF
     C> 0 GOTO 3460
3430 B=(A+PEEK(P))AND 255:SCR=RND(0)+
     PEEK(ESTIMACIONES+B)
3440 IF (B AND 240)=0 OR (B AND 15)=0 OR SCR< = HI
     GOTO 3460
3450 LP%=B:LC%=NEGRAS%:GOSUB 3890:IF LL%=0
     AND CLIB%>2 THEN HI=SCR:POSIT%=B
3460 NEXT
3470 NEXT
3480 RETURN
3490 :
3500 REM ************************************
4410 :
4420 REM RUTINA FRONTERAS
4430 FOR BX=0 TO 15
4450 BY=16*BX
4460 POKE TABLERO+BX,BV%
4470 POKE TABLERO+BY,BV%
4480 NEXT
4490 RETURN
4500 :
4510 REM ************************************
4520 REM ************* FIN DEL PROGRAMA *******
```

Amstrad CPC 464/664:

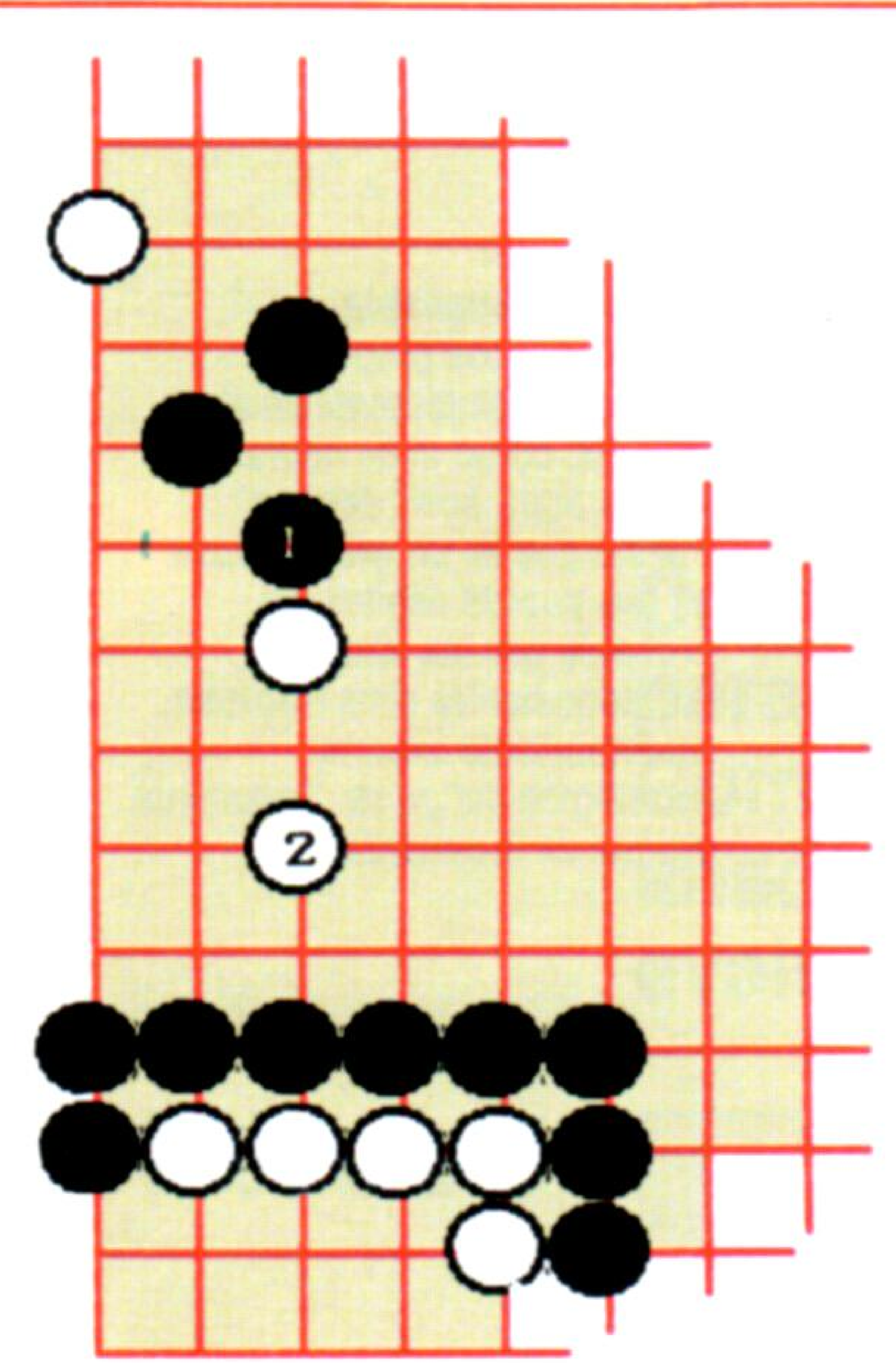
```
280 GOSUB 790: REM leer patrones
780 :
790 REM rutina lectura patrones
810 pat=tablero+&300
820 RESTORE 860
830 FOR I%=0 TO 72
840 READ p%:POKE(pat+I%),(256+p%) AND 255
850 NEXT I%
860 DATA 32,17,16,2,-15,1
870 DATA -32,-17,-16,-2,15,-1
880 DATA 33,16,17,31,16,15
890 DATA -33,-16,-17,-31,-16,-1
900 DATA -14,1,-15,18,1,17
910 DATA 14,-1,15,-18,-1,-17
920 DATA 48,33,32,48,17,16
930 DATA -48,-33,-32,-48,-17,-16
940 DATA 3,-14,2,3,-15,1
950 DATA -3,14,-2,-3,15,-1
960 DATA 64,33,32,4,-14,2
970 DATA -64,-33,-32,-4,14,-2
980 RETURN
990 :
1000 REM ************************************
2580 IF posicion%=0 THEN GOSUB 2900:T$="DEF"
2590 IF posicion%=0 THEN GOSUB 3070:T$="ATT"
2600 IF posicion%=0 THEN GOSUB 3230:T$="SAT"
2610 IF posicion%=0 THEN GOSUB 3380:T$="SCN"
2900 REM rutina conexión de defensa
2910 hi=-9999
2920 bv%=negras%:GOSUB 4420: REM frontera
2930 FOR a%=1 TO 255:dcs%=PEEK(TABLERO+a%):IF
     dcs%<>negras% THEN 3010
2940 FOR p%=pat TO pat+70 STEP 3
2950 b%=PEEK(tablero+ ((a%+PEEK(p%))AND
     255)):c%=PEEK(tablero+ ((a%+PEEK (p%+1))AND
     255))
2960 IF b%<>negras% OR c%<>blancas% THEN 3000
2970 d%=(a%+PEEK(p%+2)) AND
     255:marcador=RND(1)+ PEEK(estimaciones+d%)
2980 IF(d% AND 240)=0 OR (d% AND 15)=0 OR
     marcador<= hi THEN 3000
2990 lp%=d%:LET lc%=negras%:GOSUB 3890:IF
     ll%=0 AND clib%>2 THEN hi=marcador:
     posición%=d%
3000 NEXT p%
3010 NEXT a%
3020 bv%=0:GOSUB 4420:REM frontera
3030 RETURN
3040 :
3050 REM ************************************
3060 :
3070 REM rutina conexión de ataque
3080 hi=-9999
3090 bv%=blancas%:GOSUB 4420:REM frontera
3100 FOR a%=1 TO 255:acs%=PEEK (tablero+a%): IF
     cs%<>blancas% THEN 3180
3110 FOR p%=pat TO pat+70 STEP 3
3120 b%=PEEK(tablero+ ((a%+PEEK(p%)) AND
     255)):c%=PEEK(tablero+ ((a%+PEEK (p%+1))AND
     255))
3130 if b% <>blancas% OR c%<>negras% THEN 3170
3140 d%=(a%+PEEK(p%+2))AND 255:csr%=RND(1)+
     PEEK(estimaciones+d%)
3150 IF (d% AND 240)=0 OR (d% AND 15)=0 OR
     marcador< =hi THEN 3170
3160 lp%=d%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%> 2 THEN hi=marcador:posicion%=d%
3170 NEXT p%
3180 NEXT a%
3190 RETURN
3200 :
3210 REM ************************************
3220 :
3230 REM rutina iniciar ataque
3240 hi=-9999
3250 FOR a%=1 TO 255:sas%=PEEK(tablero+a%): IF
     sas%<>blancas% THEN 3330
3260 FOR p%=pat TO pat+70 STEP 3
3270 b%=PEEK(tablero+((a%+PEEK(p%))AND
     255)):d%=PEEK(tablero+((a%+PEEK(p%+2))AND
     255))
3280 IF b%<>blancas% OR d%<>0 THEN 3320
3290 c%=(a%+PEEK(p%+1)AND
     255:marcador=RND(1)+PEEK
     (estimaciones+c%)
3300 IF (c% AND 240)= 0 OR (c% AND 15)=0 OR
     marcador < =hi THEN 3320
3310 lp%=c%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%>2 THEN hi=marcador:posicion%=c%
3320 NEXT p%
3330 NEXT a%
3340 RETURN
3350 :
3360 REM ************************************
3370 :
3380 REM rutina inicio conexión
3390 hi=-9999
3400 FOR a%=1 TO 255:scs%=PEEK(tablero+a%): IF
     scs%<>negras% THEN 3470
3410 FOR p%=pat TO pat+70 STEP 3
3420 c%=PEEK(tablero+((a%+PEEK(p%+2))AND 255)):
     IF c% > 0 THEN 3460
3430 b%=(a%+PEEK(p%))AND 255:marcador=RND(1)+
     PEEK(estimaciones+b%)
3440 IF (b% AND 240)=0 OR (b% AND 15)=0 OR
     marcador< =hi THEN 3460
3450 lp%=b%:lc%=negras%:GOSUB 3890:IF ll%=0 AND
     clib%>2 THEN hi=marcador:posicion%=b%
3460 NEXT p%
3470 NEXT a%
3480 RETURN
3490 :
3500 REM ************************************
4410 :
4420 REM rutina frontera
4430 FOR x%=0 TO 15
4450 y%=16* x%
4460 POKE (tablero+x%),bv%
4470 POKE (tablero+y%),bv%
4480 NEXT x%
4490 RETURN
4500 :
4510 REM ************************************
4520 REM **************** fin del programa*******
```

Densidad

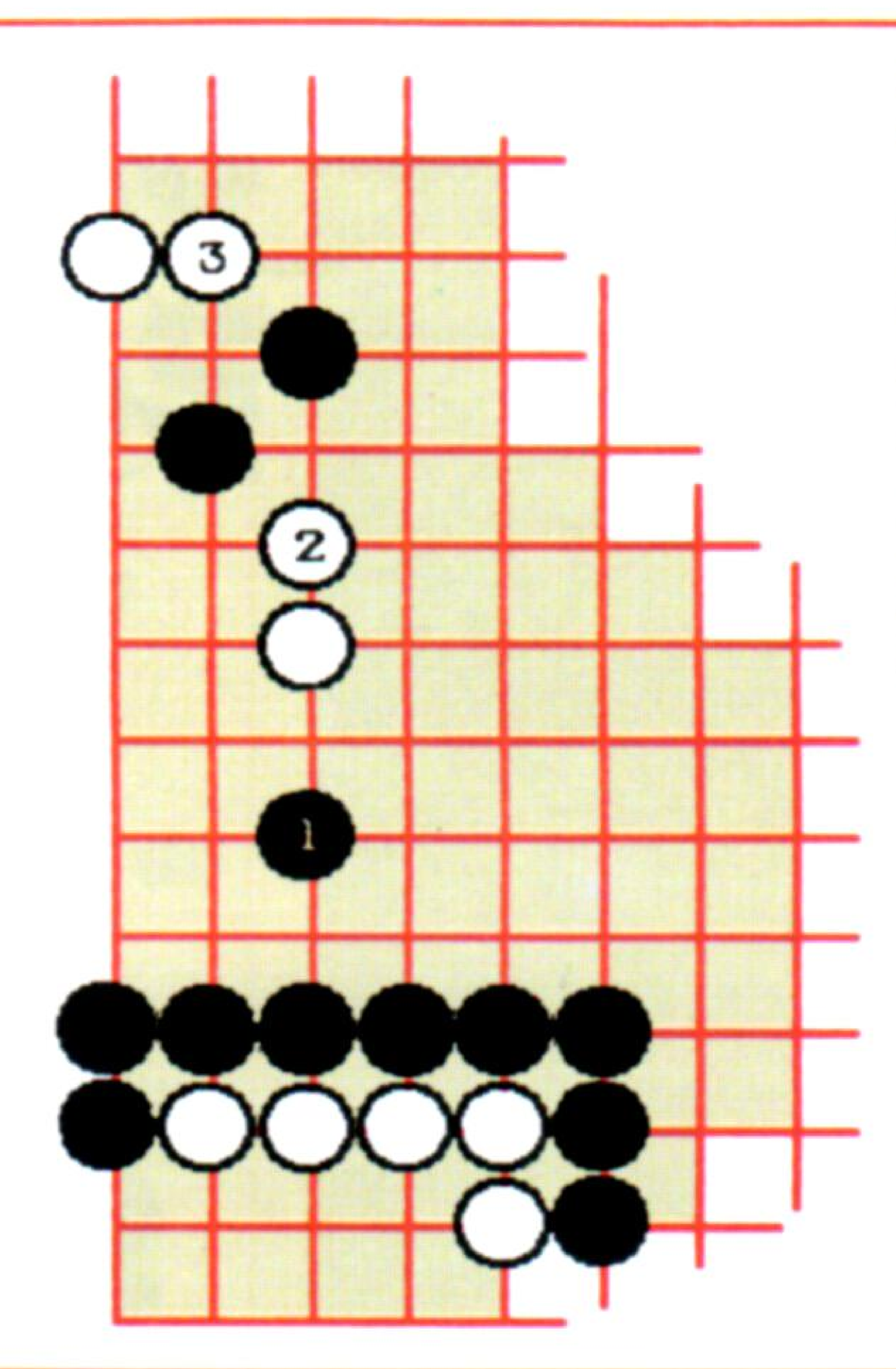


Grupo inexpugnable
Se dice que los grupos formados sólidamente en el exterior, como el de negras que vemos aquí, son "densos". Puesto que en un grupo denso no hay puntos débiles, es probable que los ataques iniciados contra ellos fracasen. Un importante axioma estratégico del *go* es: "manténte alejado de la densidad"

Caroline Clayton



Atacan las negras
Suele ser una buena estrategia utilizar los grupos densos propios como ayuda para un ataque contra fichas enemigas. En la situación que vemos aquí, la negra 1 juega encima de la ficha blanca que está sola, llevando por consiguiente a las blancas hacia el grupo denso de negras que se halla en la parte inferior del tablero. Las blancas tal vez intenten extenderse hacia afuera desde la ficha atacada jugando la ficha 2, pero es probable que a la larga no consigan el éxito, ya que el movimiento se halla más próximo a la zona densa de las negras



Otra frustración
Si las negras juegan en el lado inferior de la ficha blanca, es probable que el ataque fracase, porque las blancas pueden, simultáneamente, defenderse a sí mismas y atacar a las dos fichas negras cercanas jugando la ficha 2. Bajo la amenaza de un movimiento de las blancas en la posición 3, las negras se ven obligadas a actuar a la defensiva y el ataque se desbarata

Sinclair Spectrum:

```
280 GO SUB 790
790 REM rutina leer patrones
810 LET pat=tablero+556
820 RESTORE 860
830 FOR I=0 TO 71
840 READ p:POKE pat+I,p
850 NEXT I
860 DATA 32,17,16,2,-15,1
870 DATA -32,-17,-16,-2,15,-1
880 DATA 33,16,17,31,16,15
890 DATA -33,-16,-17,-31,-16,-15
900 DATA -14,1,-15,18,1,17
910 DATA 14,-1,15,-18,-1,-17
920 DATA 48,33,32,48,17,16
930 DATA -48,-33,-32,-48,-17,-16
940 DATA 3,-14,2,3,-15,1
950 DATA -3,14,-2,-3,15,-1
960 DATA 64,33,32,4,-14,2
970 DATA -64,-33,-32,-4,14,-2
980 RETURN
1000 REM ****************************
2580 IF posicion=0 THEN GO SUB 2900: LET
     t$="DEF"
2590 IF posicion=0 THEN GO SUB 3070: LET
     t$="ATT"
2600 IF posicion=0 THEN GO SUB 3230: LET
     t$="SAT"
2610 IF posicion=0 THEN GO SUB 3380: LET
     t$="SCN"
2900 REM rutina conexión de defensa
2910 LET hi=-9999
2920 LET bv=negras: GO SUB 4420
2930 FOR a=1 TO 255: LET
     dcs=PEEK(tablero+a):IF
     dcs%<>negras THEN GOTO 3010
2940 FOR p=pat TO pat+70 STEP 3
2950 LET b=a + PEEK p: LET c=a + PEEK
     (p+1)
2952 IF b> 255 THEN LET b=b-256: GO TO
     2952
2954 IF c> 255 THEN LET c=c-256: GO TO
     2954
2956 LET b=PEEK (tablero+b): LET c=PEEK
     (tablero+c)
2960 IF b<>negras OR c<>blancas THEN GO
     TO 3000
2970 LET d=a PEEK (p+2))
2972 IF d> 255 THEN LET d=d-256: GO TO
     2972
2974 LET marcador=RND+PEEK
     (estimaciones+d)
2980 IF INT (d/16)=0 OR d-16* INT(d/16)=0
     OR marcador<=hi THEN GO TO 3000
2990 LET lp=d: LET lc=negras: GO SUB 3890:
     IF ll=0 AND clib>2 THEN LET
     hi=marcador: LET posicion=d
3000 NEXT p
3010 NEXT a
3020 LET bv=0: GO SUB 4420
3030 RETURN
3070 REM rutina conexión de ataque
3080 LET hi=-9999
3090 LET bv=blancas: GO SUB 4420
3100 FOR a=1 TO 255: LET acs=PEEK
     (tablero+a): IF acs<>blancas THEN GO
     TO 3180
3110 FOR p=pat TO pat+70 STEP 3
3120 LET b=a+PEEK p: LET c=a + PEEK (p+1)
3122 IF b>255 THEN LET b=b-256: GO TO
     3122
3124 IF c>255 THEN LET c=c-256: GO TO
     3124
3126 LET b=PEEK (tablero+b): LET c=PEEK
     (tablero+c)
3130 IF b <>blancas OR c<>negras THEN GO
     TO 3170
3140 LET d=a+PEEK (p+2)
3142 IF d>255 THEN LET d=d-256: GO TO
     3142
3144 LET marcador=RND+PEEK
     (estimaciones+d)
3150 IF INT (d/16)=0 OR d-16* INT (d/16)=0
     OR marcador<=hi THEN GO TO 3170
3160 LET lp=d: LET lc=negras: GO SUB 3890:
     IF ll=0 AND clib> 2 THEN LET
     hi=marcador: LET posicion=d
3170 NEXT p
3180 NEXT a:LET bv=0: GO SUB 4420
3190 RETURN
3230 REM rutina inicio ataque
3240 LET hi=-9999
3250 FOR a=1 TO 255: LET sas=PEEK
     (tablero+a): IF sas<>blancas THEN GO
     TO 3330
3260 FOR p=pat TO pat+70 STEP 3
3270 LET b=a+PEEK p: LET d=a+PEEK (p+2)
3272 IF b>255 THEN LET b=b-256: GO TO
     3272
3274 IF d>255 THEN LET d=d-256: GO TO
     3274
3276 LET b=PEEK (tablero+b): LET d=PEEK
     (tablero+d)
3280 IF b<>blancas OR d<>0 THEN GO TO
     3320
3290 LET c=a+PEEK (p+1)
3292 IF c>255 THEN LET c=c-256: GO TO
     3292
3294 LET marcador=RND+PEEK
     (estimaciones+c)
3300 IF INT (c/16)=0 OR c-16*INT(c/16)=0
     OR marcador<=hi THEN GO TO 3320
3310 LET lp=c: LET lc=negras: GO SUB 3890:
     IF ll=0 AND clib>2 THEN LET
     hi=marcador: LET posicion=c
3320 NEXT p
3330 NEXT a
3340 RETURN
3380 REM rutina inicio conexión
3390 LET hi=-9999
3400 FOR a=1 TO 255: LETscs=PEEK
     (tablero+a): IF scs<>negras THEN GO
     TO 3470
3410 FOR p=pat TO pat+70 STEP 3
3420 LET c=a+PEEK (p+2)
3422 IF c>255 THEN LET c=c-256: GO TO
     3422
3424 LET c=PEEK (tablero+c): IF c> 0 THEN
     GO TO 3460
3430 LET b=a+PEEK p
3432 IF b>255 THEN LET b=b-256: GO TO
     3432
3434 LET marcador=RND+PEEK
     (estimaciones+b)
3440 IF INT (b/16)=0 OR b-16* INT (b/16)=0
     OR marcador<=hi THEN GO TO 3460
3450 LET lp=b: LET lc=negras: GO SUB 3890:
     IF ll=0 AND clib>2 THEN LET
     hi=marcador: LET posicion=b
3460 NEXT p
3470 NEXT a
3480 RETURN
4420 REM rutina fronteras
4430 FOR x=0 TO 15
4450 LET y=16* x
4460 POKE tablero+x,bv
4470 POKE tablero+y,bv
4480 NEXT x
4490 RETURN
4500 :
4510 REM ****************************
4520 REM **** FIN DEL PROGRAMA *******
```



Ahorro razonable

Centramos nuestra atención en las definiciones de dos puntos y los métodos de compilación

Una facilidad importante del FORTH es que sus definiciones de dos puntos son semicompiladas. Si bien no se convierten en código máquina (como en una compilación completa), se procesan para que ocupen menos espacio y se ejecuten más rápidamente. Como resultado de ello, los programas en FORTH son muy eficientes.

Existen varios métodos de semicompilación. El método figFORTH es, con mucho, el más corriente, al punto de que otros métodos se consideran no ortodoxos. Supongamos que usted define una palabra:

```
:CUADRADO (n—n* n) DUP*;
```

En primer lugar, se coloca un *encabezador* en el diccionario. Éste contiene (el verdadero orden de entrada puede variar entre las distintas versiones del FORTH) el nombre CUADRADO y la dirección de memoria del encabezador para la palabra definida previamente. De este modo todos los encabezadores se unen entre sí y, comenzando por el más nuevo, el ordenador puede ir buscando hacia atrás a través de todo el diccionario hasta hallar la palabra que esté buscando. Estas dos partes del encabezador se denominan *campo del nombre* y *campo de unión*.

Tras el encabezador viene la definición. Primero hay un código que muestra de qué tipo de definición se trata; hay códigos diferentes para definiciones de dos puntos (como en este caso), variables, constantes, las estructuras preparadas por CREATE, palabras definidas en código máquina para el sistema y para cualquier otra clase de palabras. Esta parte se denomina *campo del código*.

A continuación viene la parte principal de la definición de dos puntos. Ésta es una lista de las direcciones de las palabras que utiliza (en realidad son las direcciones de los campos del código de esas palabras, también conocidas como sus *direcciones de compilación*). Esta parte principal se denomina *campo de parámetros*.

Usted termina con lo siguiente:

campo del nombre:	"CUADRADO"
campo de unión:	dir. del encabezador de la pal. anterior (2 bytes)
campo del código:	cód. para una definición de dos puntos (2 bytes)
campo de paráms.:	dir. de compilación de DUP (2 bytes)
	dir. de compilación de* (2 bytes)
	dir. de compilación de ; (2 bytes)

Ahora puede empezar a ver cómo se compilan las definiciones de dos puntos. Primero se establecen los campos del nombre y de unión. Esto es igual para cualquier definición de palabra nueva. A continuación, : coloca el código especial para las definiciones de dos puntos, y también coloca al FORTH en un estado de compilación especial. A partir de este momento, cuando entre una palabra, ésta no se ejecuta de la forma normal, sino que se incluye en el diccionario su dirección de compilación. También se suprime el mensaje OK para hacerle saber qué está sucediendo.

Esto continúa hasta ;, de modo que, obviamente, ; ha de ser especial. No sólo se incluye en el diccionario, sino que también tiene el efecto inmediato de decirle al FORTH que vuelva del estado de compilación a su estado normal. Palabras como ésta, que se deben ejecutar directamente como parte del proceso de compilación, se denominan palabras *inmediatas*. No se incluyen en el diccionario a menos que ellas mismas se ocupen de ello. En realidad, lo que ; incluye en el diccionario no es su propia dirección de compilación, sino la de una sombra anónima, la acción en tiempo de ejecución de ;.

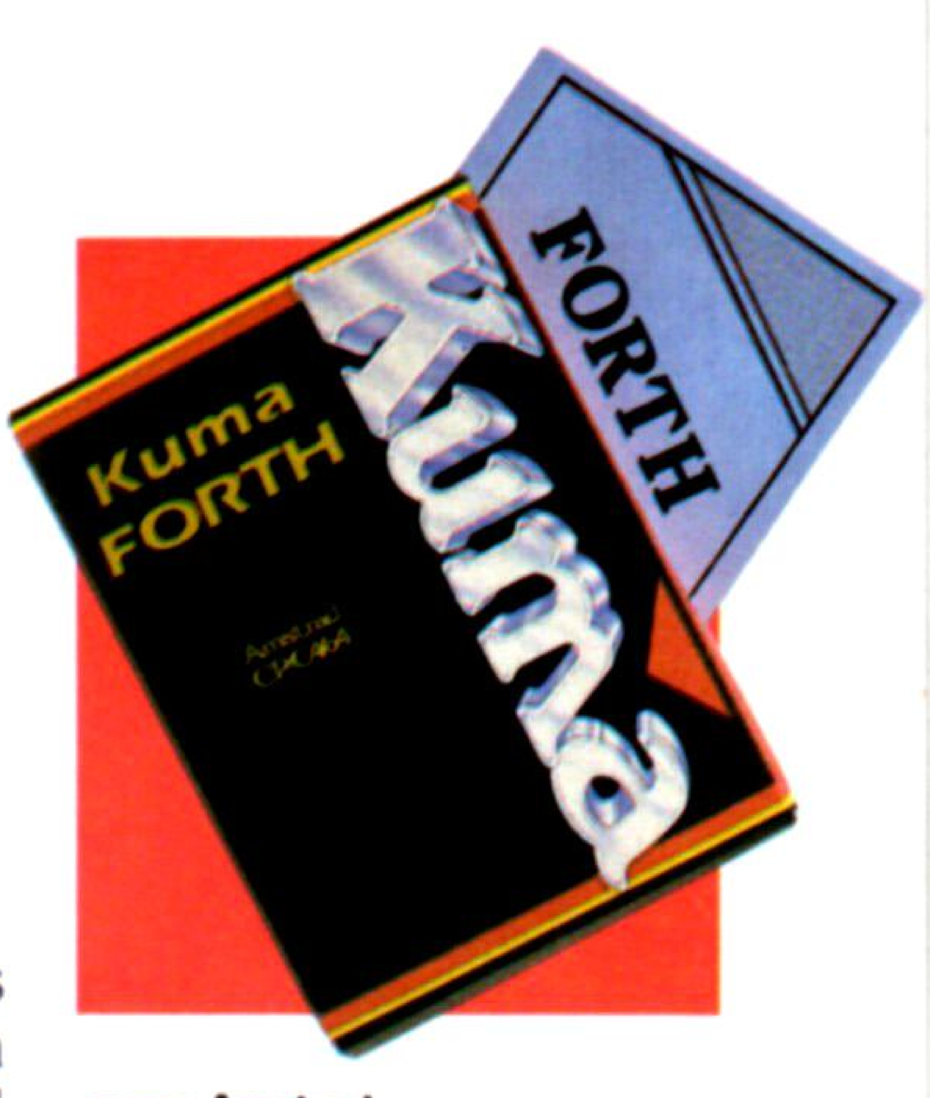
Cuando se ejecuta CUADRADO (p. ej., si usted entra 4 CUADRADO .), el FORTH lo busca en el diccionario y encuentra su definición. Ésta le dice entonces, a su vez, que ejecute DUP y * y luego se detenga, pero no necesita buscarlas en el diccionario porque ya posee las direcciones de sus definiciones. En consecuencia, las definiciones de dos puntos se pueden ejecutar bastante velozmente.

Esto describe las definiciones de dos puntos más simples. Tanto los números como las estructuras de programa presentan dificultades.

Está claro que los números no poseen direcciones de compilación, de modo que se compilan de una forma compuesta: primero, la dirección de compilación de un *manipulador literal* (otra sombra anónima) y luego el número propiamente dicho. Cuando se ejecuta desde el interior de una definición de dos puntos, el efecto del manipulador lite-

Campos de código

Los campos de código son bastante inteligentes. El número de código en realidad es la dirección de memoria del código máquina que se ejecuta cada vez que se ejecuta la palabra. Para cualquier definición de dos puntos, el código máquina recuerda a dónde ha de retornar el intérprete de FORTH cuando termine esta definición en particular. Esto se efectúa colocando la dirección de retorno en otra pila, la *pila de retorno*, y volviendo a dirigir el intérprete hasta la definición actual. Hay rutinas diferentes en código máquina para constantes y variables (que en sus campos de parámetros poseen un único número). Las palabras *primitivas*, como +y*, están escritas en código máquina. Éste se almacena en el campo de parámetros y el campo de código contiene su dirección



FORTH Amstrad
El Amstrad CPC 464/664 se une a otros micros personales en la capacidad para ejecutar FORTH en virtud de este paquete de Kuma Software. La versión Kuma del FORTH es una implementación de figFORTH, con facilidades para punto flotante, series y gráficos en color

ral consiste en decirle al sistema que "los dos bytes siguientes que ves no son una dirección de compilación a ejecutar, sino un número a colocar en la pila". ."posee un compuesto similar, formado por un manipulador seguido por la serie propiamente dicha.

La siguiente dificultad se plantea con estructuras de programación tales como IF...ELSE...THEN. Si usted estuviera programando en código máquina (o en los BASIC más primitivos), tendría que hacer esto con saltos, algo así como:

```
1000 Si parte superior pila =0 THEN GOTO 1100
1010 REM hacer esta parte si la condición es falsa
     :
     :
1090 GOTO 1200
1100 REM hacer esta parte si la condición es
     verdadera
     :
     :
1200 REM continuar a partir de aquí en ambos casos
```

El FORTH compilado es bastante parecido a esto. En FORTH hay dos palabras, BRANCH (salto incondicional) y ?BRANCH (quitar la parte superior de la pila y saltar si es cero), pero usted no puede emplearlas en el curso normal de los acontecimientos porque hay que utilizarlas como formas compuestas (como un número compilado): la dirección de compilación seguida por la dirección del lugar hasta donde saltar. Es labor de IF, ELSE, THEN, etc., que son palabras inmediatas, el calcular estas direcciones de salto y compilar las formas compuestas. Por ejemplo, IF incluye la dirección de compilación de ?BRANCH y espacio para la dirección del salto, pero no puede llenar la dirección del salto porque aún no sabe dónde está ELSE. En cambio, deja en la pila la dirección de este espacio para que ELSE pueda llenar la dirección de salto. En realidad, IF también deja en la pila un número de comprobación. Cada clase de estructura posee su propio número de comprobación, de modo que si ELSE no encuentra a IF, entonces sabe que algo anda mal: o que nunca hubo un IF, o bien que algo de en medio se ha mezclado en la pila. Ello impide que se escriban cosas como:

```
IF...BEGIN...ELSE...UNTIL...THEN
```

Veamos algunos ejemplos.

1. Supongamos que su versión de FORTH distingue entre caracteres en mayúscula y en minúscula, y que todas las palabras estándares se entran en el diccionario en mayúsculas. Esto significaría que no se reconocería loop, mientras que sí se reconocería LOOP. Si usted quisiera entrar palabras tanto en minúsculas como en mayúsculas, podría comenzar por:

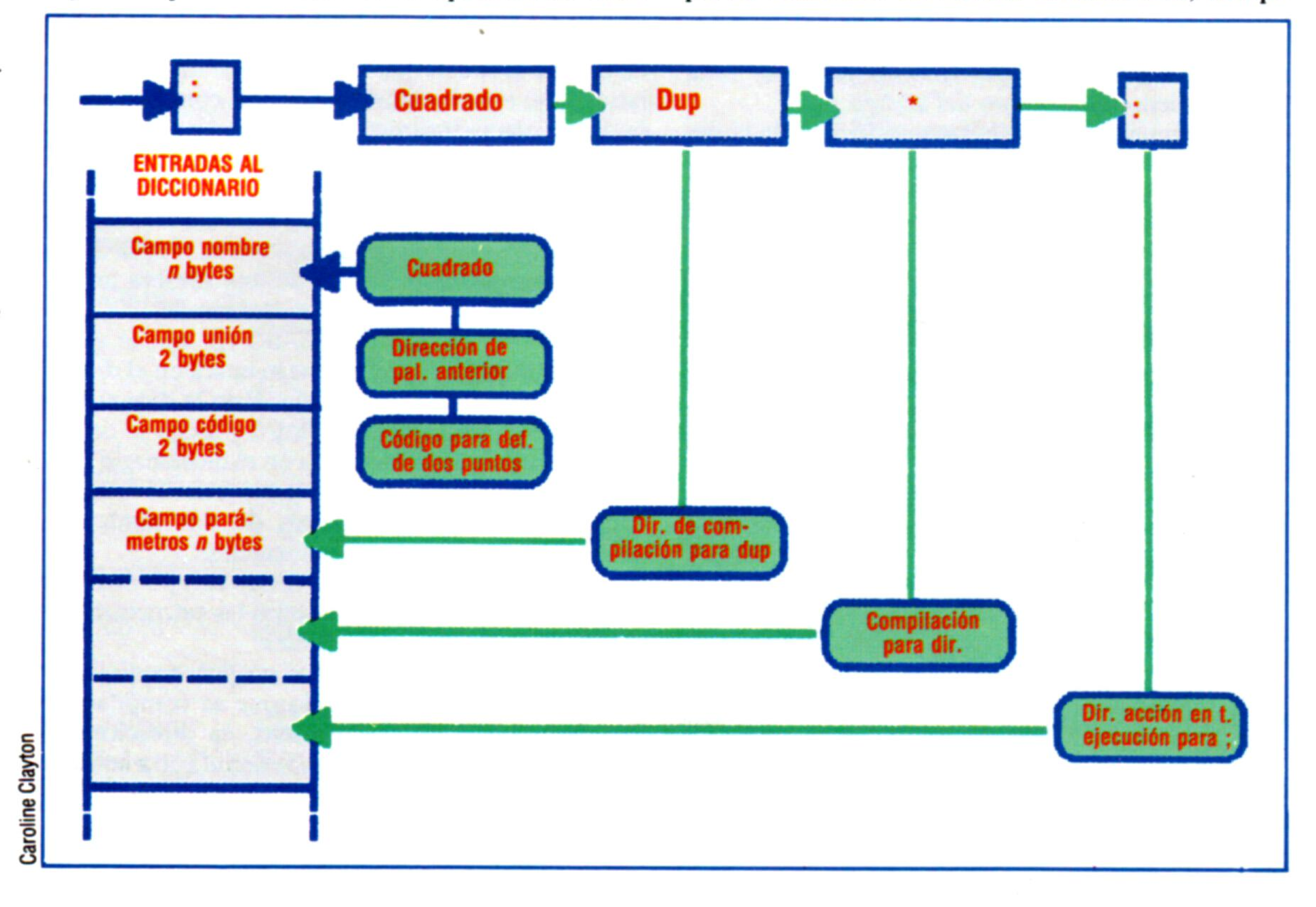
```
:drop DROP;
:roll ROLL;
:variable VARIABLE (sí, esto funcionaría);
```

Tendría problemas con:

```
:loop LOOP (esto no funcionaría);


:loop [COMPILE] LOOP;IMMEDIATE
```

Usted habrá de tener una mente muy lúcida para ver lo que hace esto, puesto que hay tres circunstancias distintas a tener en cuenta. Primero debe considerar qué sucede cuando se define loop. Aunque LOOP es inmediata, [COMPILE] hace caso omiso de ella, de modo que la dirección de compilación de LOOP pasa a la definición de loop. La segunda circunstancia, tiempo de compilación, es cuando se utiliza loop para redefinir otra palabra. Dado que es inmediata, se ejecuta directamente y tiene por efecto ejecutar LOOP. Éste es el momento que exige mayor concentración, porque es cuando loop cumple su cometido. La tercera circunstancia, tiempo



Almacenamiento de palabras
El FORTH emplea un método de "semicompilación" para facilitar la ejecución de los programas. Una vez entrada una definición de dos puntos, se establecen en el diccionario diferentes campos, comenzando con los campos de nombre, de unión y de código. El FORTH entra, entonces, en una *modalidad de compilación*, en la que la palabra no se entra directamente en el diccionario, sino que es su *dirección de compilación* la que se entra en el campo de parámetros de la definición. Tras encontrar el ; el FORTH vuelve a la *modalidad de ejecución*



Palabras importantes

[y] le permite pasar temporalmente al estado de ejecución estando aún en medio de una definición de dos puntos.

LITERAL, una palabra inmediata, quita la parte superior de la pila y la compila en una definición de dos puntos de la forma normal para un número. Típicamente usted la utilizaría con [y] para evaluar una expresión en tiempo de compilación y compilar el resultado. Por ejemplo:

```
[9 16*] LITERAL
```

en una definición de dos puntos tiene el mismo efecto que 144.

[COMPILE] es una palabra inmediata y dice que la siguiente palabra no se debe ejecutar sino incluir, aun cuando sea inmediata.

COMPILE no es inmediata. Al ejecutarse, incluye en el diccionario la siguiente palabra.

BRANCH y ?BRANCH se utilizan como en el ejemplo cuando usted construye saltos. > MARK y > RESOLVE se utilizan para llenar la dirección de salto para un salto hacia adelante: > MARK se utiliza en el salto y > RESOLVE en su destino. < MARK y < RESOLVE son similares para saltos hacia atrás, con < MARCK en el destino y < RESOLVE en el salto.

IMMEDIATE hace inmediata la palabra anterior.

de ejecución, es cuando se ejecuta la otra palabra, y después entra en vigor lo que fuere que haya compilado LOOP en su definición.

2. He aquí un par de palabras, ?DO y ?LOOP. Son como DO y LOOP, pero si el inicio ya es tan grande como el límite, entonces el bucle no se ejecuta nunca.

```
?DO equivale a      OVER OVER > IF DO
?LOOP equivale a    LOOP ELSE DROP DROP
                    THEN
```

Sin embargo, si usted intenta definir:

```
:?DO (no está bien) OVER OVER > IF DO;
```

tendrá el mismo problema que con loop. La definición correcta es:

```
:?DO
   COMPILE OVER
   COMPILE OVER
   COMPILE >
   [COMPILE] IF
   [COMPILE] DO
;IMMEDIATE

:?LOOP
   [COMPILE] LOOP
   [COMPILE] ELSE
   COMPILE DROP
   COMPILE DROP
   [COMPILE] THEN
;IMMEDIATE
```

3. Las estructuras de caso proporcionan una gama de acciones según cual sea el valor de algo. No se incluyen en el estándar, pero hay una utilizada comúnmente que puede definir por sí mismo. Su forma es:

```
valor CASE
1.ª posibilidad OF...ENDOF
2.ª posibilidad OF...ENDOF
3.ª posibilidad OF...ENDOF
...

Qué hacer si no responde a ninguna posibilidad

ENDCASE
```

Por ejemplo, el valor podría ser el código ASCII de una tecla pulsada y usted puede especificar distintas acciones para los diferentes caracteres que fueran factibles.

Si observa los saltos necesarios para hacer esto, verá que cada OF necesita un salto condicionado a su ENDOF, y que cada ENDOF necesita un salto a ENDCASE. ENDCASE podría necesitar llenar, o *resolver*, varias direcciones de salto. En FORTH no hay ninguna palabra estándar para hacer esto, de modo que está obligado a calcular por sí mismo los saltos para CASE, lo que hará mediante >MARK y >RESOLVE. >MARK apila la dirección del espacio para una dirección de salto, como hace IF, y >RESOLVE lo quita de la pila y coloca el espacio.

```
:CASE  (run: valor —valor
        (compile: —0)
0       (cantidad de OFs—ninguno hasta ahora)
;IMMEDIATE

:OF     (run:valor, posibilidad— [si pareja])
 (     valor, posibilidad—valor [ninguna pareja])
        (compile:—marca para salto a ENDOF)
COMPILE OVER
COMPILE=
COMPILE?BRANCH>MARK
COMPILE DROP (drop valor si pareja)
;IMMEDIATE

:ENDOF (run:—)
   (compile: OF cuenta, marca del OF—
   (              marca para salto a ENDCASE,
                  cuenta + 1 de OF)
COMPILE BRANCH > MARK
SWAP > RESOLVE (resolver salto desde OF)
SWAP 1 +            (incrementar cuenta OF)
;IMMEDIATE
:ENDCASE  (run:valor—
           (compile:marcas desde los ENDOF,
           OF
           cuenta—)
   COMPILE DROP
   0?DO
     > RESOLVE
   ?LOOP
;IMMEDIATE
```

Aunque esto es complicado, el mero hecho de que usted pueda hacerlo muestra lo ampliable que es el FORTH.

En la práctica, también debería utilizar números de comprobación para asegurar que una estructura CASE se ensamble de la forma adecuada.

El FORTH-79 carece de BRANCH, >MARK, >RESOLVE, etc., y aunque el figFORTH posee palabras similares, usted tendrá que ahondar mucho más para utilizarlas.

Amstrad para lo que gusten

Al comenzar esta serie sobre el OS del Amstrad CPC 464 y 664 examinaremos los mapas de la ROM y de la RAM

Tanto en el Amstrad CPC 464 como en el 664 queda muy poco de la más reciente tecnología del silicio. Sin embargo, las configuraciones que los ingenieros informáticos suelen implementar en el hardware se compensan en las máquinas Amstrad con un sistema operativo más globalizador. De aquí que se necesitara una cantidad mínima de hardware para proporcionar las facilidades disponibles en las máquinas Amstrad, lo que dio como resultado un micro menos caro.

El CPC 464 emplea una CPU Z80A, acompañada de 64 K de RAM, 32 K de ROM, un PIA 8255 (adaptador de interfaces periféricas), un generador de sonido AY-3-8912, un CRTC 6845 (*cathode-ray tube controller:* controlador de tubo de rayos catódicos) y una matriz de puertas fabricada a medida para el cliente (ULA: *uncomitted logic array:* disposición lógica no comprometida).

El sistema operativo, conocido como *firmware,* ocupa la mitad inferior de la ROM de 32 K. El resto de la ROM contiene la versión del BASIC del Amstrad, denominada Locomotive BASIC. El firmware se divide en subsecciones, cada una de las cuales se encarga de un aspecto particular del sistema. Esto ofrece una cómoda estructuración que sirve de base para nuestro estudio del OS del Amstrad que ahora iniciamos. De momento echemos una mirada general al sistema.

El mapa de memoria del sistema queda reflejado en el esquema (abajo), donde aparecen las dos mitades de la ROM como dos ROM separadas lógicamente y con 16 K cada una, ambas conectables por separado mediante la ULA. Se observará que toda la ROM de 32 K se superpone a los primeros y últimos 16 K de la RAM que quedan siempre libres para lectura y escritura. La ULA ofrece también la facilidad para conectar la ROM superior con cualquiera de las 252 ROM laterales de ampliación. En próximos capítulos abordaremos el empleo de las ROM de ampliación.

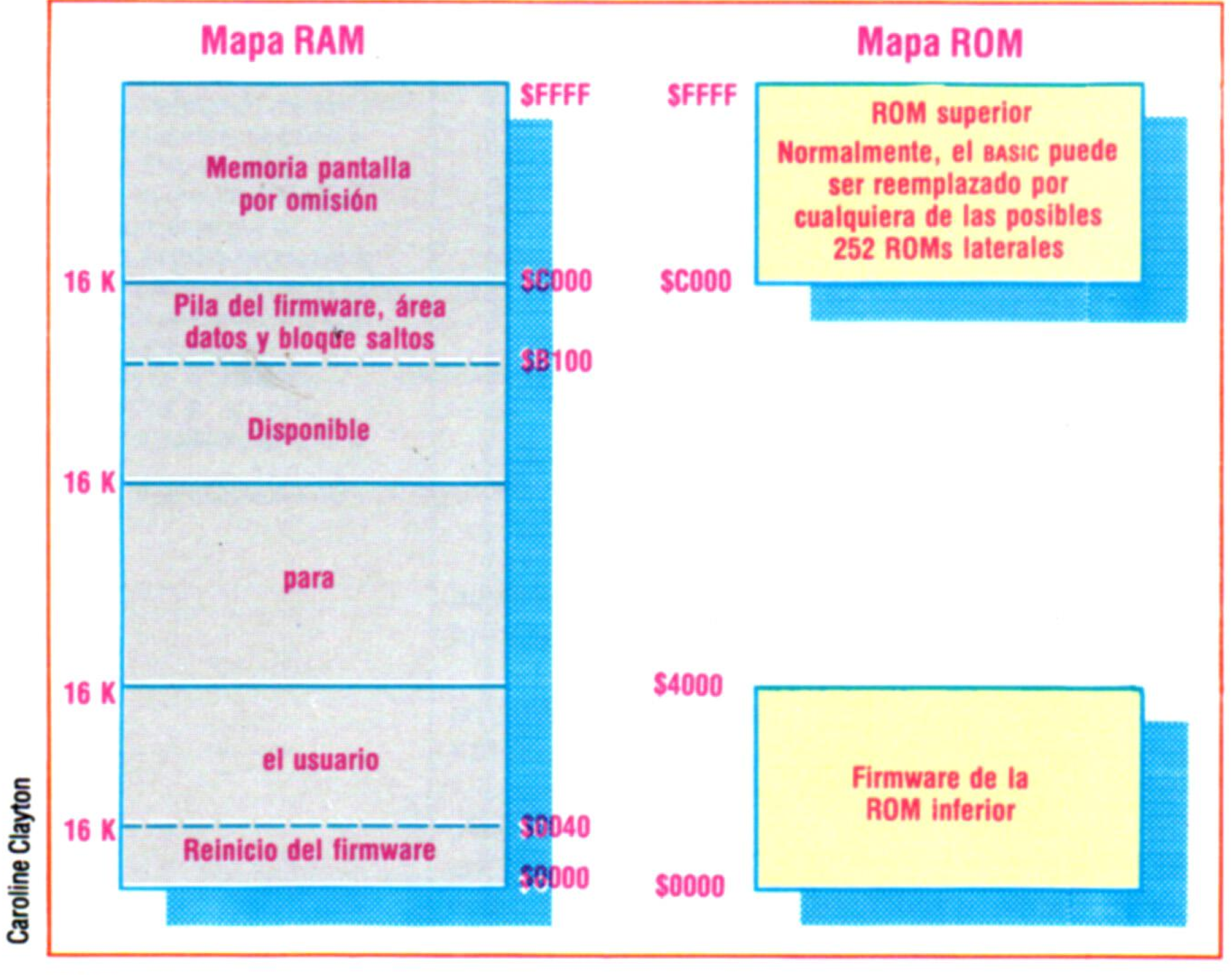
La pantalla es controlada por el chip 6845 y requiere 15 K de RAM, que la memoria ofrece por lo general inmediatamente debajo de la ROM superior. No obstante, es posible elegir cualquiera de los cuatro bloques de 16 K para que alberguen los datos de pantalla (lo que permite la conexión de pantallas múltiples a velocidades extraordinarias).

Cuando se ha conectado la ROM superior (p. ej., cuando el BASIC está bajo control) todo lo que se lee de los 16 K superiores de la memoria es organizado por la ULA para que dé posiciones dentro de la ROM. Cualquier cosa escrita en este bloque de 16 K pasa a través de la ROM hasta la RAM subyacente, que habitualmente se encuentra ocupada por la pantalla. La lectura desde la RAM requiere que esté desconectada la ROM superior; el firmware proporciona diversos modos de hacerlo.

Un acceso a la memoria similar se aplica a la ROM inferior que contiene el firmware. Pero ¿qué sucede si la ROM inferior está desconectada? ¿Cómo pueden ejecutarse los programas del sistema operativo si han desaparecido? Este problema se resuelve no accediendo directamente a la ROM inferior a través de programas, haciendo en su lugar todas las llamadas al sistema operativo a través de un *bloque de saltos* (*jumpblock*). Se trata de un área de memoria que tiene una posición fijada y contiene una serie de entradas que sencillamente realizan un salto (o acción equivalente) a distintas rutinas dentro del firmware. El empleo de un bloque de saltos permite el cambio de las direcciones actuales de las rutinas del firmware, sin afectar los programas existentes. Conviene llamar las rutinas del firmware a través del bloque de saltos.

El bloque de saltos reside en la RAM de sistema en el tercero de los bloques de 16 K, por lo que no se puede desconectar igual que la ROM. Se ha previsto la imposibilidad de que el usuario sobreescriba esta área. Aun así, hay situaciones en las que es deseable alterar el bloque de saltos para eliminar o sustituir parte del sistema operativo. Esto se denomina *parcheo del bloque de saltos (jumpblock patching)* que analizaremos con detenimiento en próximos capítulos.


La ROM encima de la RAM
El mapa de memoria del Amstrad muestra claramente cómo el chip de ROM de 32 K "despagina" los 16 K superiores e inferiores de la RAM. El firmware puede emplearse para activar y desactivar ambas páginas de RAM, así como para comprobar la presencia de hasta 252 ROMs laterales. Los datos de pantalla pueden contenerse en cualquiera de los cuatro bloques RAM de 16 K



Caroline Clayton

Cómo trabaja el bloque de saltos

Todos los puntos de entrada al sistema operativo tienen una entrada correspondiente en el bloque de saltos. Cuando el usuario desea acceder al sistema operativo, se hace una llamada a la entrada adecuada del bloque de saltos. Esta entrada emite, entonces, una instrucción RST, la cual, a su vez, llama al firmware. Cada entrada tiene tres bytes de longitud; en la puesta en marcha cada una contiene una instrucción RST seguida de una dirección de entrada del sistema operativo.

Cuando no es necesaria, es posible desconectar la ROM inferior, por cuya razón se mantiene una copia de las rutinas de reiniciación en la RAM contigua a la ROM. Estas rutinas de reiniciación activan la ROM del firmware antes de ser llamada a continuación. Cuando la rutina del sistema operativo retorna, las ROM son restauradas al estado que tenían a la entrada. Esto resuelve el dilema de decidir qué ROM hay que conectar antes de llamar al sistema operativo. La mayoría de las instrucciones de reiniciación se emplean para disponer de un conjunto ampliado de instrucciones; las instrucciones adicionales son principalmente para controlar la activación de la ROM. En la página siguiente se proporciona un cuadro con estas instrucciones de reiniciación y sus empleos.

El sistema de bloque de saltos permite reemplazar las entradas existentes del sistema operativo por rutinas sustitutas. Por ejemplo, si desea escribir sus propias rutinas de manejo de la pantalla, éstas pueden "parchearse" de modo que sustituyan las rutinas existentes alterando la entrada del bloque de saltos para que apunte a una nueva rutina. Damos dos diagramas para mostrar la diferencia entre el flujo de control cuando se pasa una instrucción al sistema operativo de la manera habitual, y la misma cadena cuando ha sido interceptada por el usuario.

Este procedimiento de parchear un bloque de saltos es de hecho el método empleado tanto en el CPC 464 como en el 664 para conectar entre las instrucciones de cassette y de disco.

Rutina ilustrativa

Un ejemplo de tal método lo tendría si supone que ha escrito una rutina llamada SEND SCREEN, que envía texto a la pantalla, y desea usar ésta en lugar de la rutina existente del sistema operativo. La entrada en el bloque de saltos para imprimir caracteres en la pantalla se llama TXT WR CHAR y tiene dirección en 0BB5DH. El acoplamiento se logra sustituyendo un salto a la nueva rutina en el bloque de saltos.

El procedimiento adoptado antes funciona bien cuando se ha escrito una rutina enteramente nueva y la entrada no se necesita. El problema surge cuando bien se está todavía empleando la entrada original o bien se necesita una llamada desde dentro de la nueva rutina a otra entrada en el bloque de saltos. Pongamos este ejemplo de programa en BASIC que sólo toma códigos de caracteres y los envía a la pantalla:

```
10 MODE 1          limpia pantalla y envía
                   cursor a inicio
```

Control del programa

Salta y brinca
El bloque de saltos permite al usuario emplear el firmware de una manera completa y sin complicaciones, y también redirigir las llamadas del sistema hacia rutinas definidas por el usuario. Este último proceso implica el parcheado del bloque de saltos, y los diagramas que aquí exponemos muestran las distintas fases necesarias, según el método empleado

Caroline Clayton

```
20 WHILE-1                   establece un bucle infinito
30 INPUT "Codigo             toma el codigo de caracter
   de caracter";C;
40 IF C <0 OR C>255          comprueba validez codigo
   then ?:GOTO 30
50 ?CHR$(C);                 lo envía
60 WEND                      itera el bucle
```

El programa permite imprimir todos los caracteres de gráficos y códigos de control. Sin embargo, hay un código que desactiva la pantalla, lo que significa que el programa quedaría "colgado". Por consiguiente, se necesita establecer un test para impedir el envío de este código. Este test podría insertarse fácilmente en la línea 40, pero lo estableceremos en la 50 para favorecer el propósito general de este ejemplo (línea en la que el carácter se imprime efectivamente). Para ello, hay que hacer el test antes de que el carácter sea enviado a la pantalla, empleando la entrada del bloque de saltos TXT OUTPUT en la dirección $BB5A. La entrada original requiere que se pase al registro A el caracter que ha de ser enviado y que se guarden todos los registros a la salida.

El modo más sencillo de lograr el parcheado *(patch)* del bloque de saltos es el que muestra el siguiente listado.

```
        txt_ou:   equ   #BB5A              ;rutina salida bloque
        disabl:   equ   21                 ;caracter desactivador pantalla
        ;Sigue nueva rutina - el caracter a visualizar esta en A
FE15    new;      cp    disable            ;es desactivado VDU?
C8                ret   z                  ;ignorarlo, si es así
000000  entry:    defb  0,0,0              ;inserta entrada original
        ;La direccion de retorno esta ya en la pila a la entrada
        ;a la rutina, luego es innecesaria una instruccion RET
        ;La entrada del bloque de saltos existente en txt out debe copiarse en ENTRY
        ;La entrada original JMP debe sustituirse por NEW
```

El enfoque adoptado funcionará con cualquiera de las entradas de rutina en el bloque de saltos. Sin embargo, habrá de tener cuidado cuando se está parcheando una rutina y asegurarse de que nadie lo haya hecho antes. El problema está en que algunas entradas del bloque de saltos cuentan con su dirección para funcionar adecuadamente (es decir, si se copia la entrada en otra parte, cuando sea llamada no funcionará bien). Para parchear el bloque con este tipo de entradas se adoptará otro enfoque diferente.

Proporcionamos un diagrama que muestra un método alternativo de parchear el bloque de saltos. Se hace una copia de la entrada actual y se sustituye en su lugar por un salto a la nueva rutina. Ésta copia, entonces, la antigua entrada en su lugar apropiado dentro del bloque de saltos y la llama. Antes del retorno, el salto a la nueva rutina se vuelve a colocar en el bloque de saltos.

Este segundo listado en código máquina es un programa que toma nuestro ejemplo y lo parchea de la manera ya descrita.

```
        txt_ou:   equ   #BB5D              ;rutina salida del bloque saltos
        disabl:   equ   21                 ;caracter desact. pantalla
FE15    new:      cp    disable            ;punto entrada rutina
C8                ret   z                  ;ignora desactiv.
;ahora conecta entrada antigua al bloque saltos y la llama
CD2529            call  copy               ;conecta entrada
CD5DBB            call  txt out            ;envia caracter
CD2529            call  copy               ;recopia la entrada
C9                ret

F5      copy:     push  af                 ;debe guardar todos los regs
C5                push  bc
D5                push  de
E5                push  hl
0603              ld    b,3                ;ningun byte en entrada
215DBB            ld    hl,tx_out          ;entrada antigua
113F29  cloop:    ld    de,txt_cpy         ;copia area
4E                ld    c,(hl)             ;lee entrada actual
1A                ld    a,(de)             ;y la reemplaza
77                ld    (hl),a             ;por su sustituta
79                ld    a,c
12                ld    (de),a             ;y guarda la antigua
23                inc   hl                 ;apunta al byte siguiente
13                inc   de
10F7              djnz  cloop              ;ejecuta los tres
E1                pop   hl
D1                pop   de
C1                pop   bc
F1                pop   af
C9                ret
   ;almacenamiento de nueva entrada bloque saltos hasta que se necesite
C3      txt_cp:   defb  #c3                ;codigo de bloque saltos
1829              defw  new                ;para rutina sustituta
```

Este enfoque a primera vista puede parecer complicado, pero su perfecto funcionamiento está garantizado sea cual fuere el contenido original del bloque de saltos.

Es importante darse cuenta de que el firmware debe ver toda rutina sustituta como si fuera una rutina por omisión (esto es, todos los parámetros pasados y devueltos deben tener la misma definición que los de la rutina sustituida). En el ejemplo que hemos proporcionado, esto se consigue sencillamente guardando los registros devueltos mientras se efectúa la copia.

Merece la pena mencionar aquí la razón para colocar las nuevas rutinas por debajo del HIMEM. La disposición de la RAM de sistema ya quedó descrita, y se mostró que la gran área de RAM libre que está en los tres bloques de 16 K de la parte inferior está a disposición del BASIC. De hecho, el límite inferior está fijado (aunque más adelante mostraremos cómo se puede reservar memoria debajo del área operativa del BASIC), y el límite superior se establece con el valor de HIMEM. En realidad, un programa BASIC "crece" a partir del límite inferior hacia arriba hasta HIMEM.

Para reservar espacio para una rutina en código máquina, debemos asegurarnos de que el BASIC no está dispuesto para acceder a la misma área, y por lo dicho anteriormente está claro que esto se consigue bajando el HIMEM. Para que este proceso sea lo más económico posible en espacio de memoria que queda para el BASIC, el HIMEM bajará sólo la longitud de la rutina que deseamos emplear (más su área de datos) y la rutina se cargará en la RAM que ha quedado libre.

Instrucciones de reinicio
El área de memoria entre #0000 y #003F está duplicada en la RAM y en la ROM, de modo que las instrucciones RST pueden ejecutarse con independencia de que la ROM inferior esté activada o desactivada. El Z80 proporciona ocho instrucciones de reinicio, muchas de las cuales son de uso exclusivo del Amstrad para la administración de la memoria. He aquí la lista de sus funciones

Dirección en hexa		Función
0000	RST 0	Restaura (como en la puesta en marcha)
0008	RST 1	Salta a la rutina de los 16 K inferiores. Los dos bytes que siguen la RST se consideran como una dirección de 14 bits (0000 a 3FFF), con los bits 15 y 14 que señalan el estado de activación de la ROM –bit 15 activado = ROM superior desactivada; bit 14 activado = ROM inferior desactivada
0010	RST 2	Llamada lateral - RST seguida de dos bytes; los bits 15 y 14 indican una de las cuatro ROM laterales; los bits del 0 al 13 dan el desplazamiento a añadir a #C000 para obtener la dirección de la rutina
0018	RST 3	Llamada de lejos - llama cualquier rutina en la ROM o RAM. RST + dos bytes que apuntan a la "dirección lejana" de 3 bytes. Bytes dirección lejana 0/1 = dirección; byte 2 = byte de selección de ROM así: #00–#FB = número de ROM lateral, activa la superior, desactiva la inferior; #FC = activación superior e inferior; #FD = activa superior, desactiva inferior ; #FE = desactiva superior, activa inferior ; #FF desactiva superior e inferior
0020	RST 4	Lee byte de RAM apuntado por HL - desactiva ROM superior, activa inferior. Al retorno A retiene el byte leído
0028	RST 5	Llamada del firmware - activa ROM inferior y salta a rutina apuntada por los dos bytes que siguen a RST
0030	RST 6	Reinicio del usuario. Se dejan sin usar los bytes #30 al #37
0038	RST 7	Entrada de interrupción. Se deja solo lo mejor. Si usted desea emplear las interrupciones del modo 1 habrá de considerar el empleo de las facilidades de "eventos" del firmware que analizaremos más adelante en este curso

Editorial Delta, S.A.



9 788485 822836
10101